x

Singular

A Computer Algebra System for Polynomial Computations

Manual

Version 4.3.2, 2023

Singular is created and its development is directed and coordinated by
W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann

Principal developers:
O. Bachmann, M. Brickenstein, W. Decker, A. Frühbis-Krüger, K. Krüger,
V. Levandovskyy, C. Lossen, W. Neumann, W. Pohl, J. Schmidt, M. Schulze,
T. Siebert, R. Stobbe, E. Westenberger, T. Wichmann, O. Wienand

Department of Mathematics
Centre of computer algebra
University Kaiserslautern-Landau
D-67653 Kaiserslautern

Short Contents

1 Preface

SINGULAR version 4.3.2
University of Kaiserslautern-Landau
Department of Mathematics and Centre for Computer Algebra
Authors: W. Decker, G.-M. Greuel, G. Pfister, H. Schoenemann
Copyright © 1986-2023

NOTICE

Some single files have a copyright given within the file: Singular/links/ndbm.* (BSD)

The following software modules shipped with SINGULAR have their own copyright: the omalloc library, the readline library, the GNU Multiple Precision Library (GMP), NTL: A Library for doing Number Theory (NTL), Flint: Fast Library for Number Theory, the Singular-Factory library, the Singular-Factory library, the Singular-libfac library, surfex, and, for the Windows distributions, the Cygwin DLL and the Cygwin tools (Cygwin), and the XEmacs editor (XEmacs).

Their copyrights and licenses can be found in the accompanying files COPYING which are distributed along with these packages. (Since version 3-0-3 of SINGULAR, all parts have GPL or LGPL as (one of) their licences.)

Please send any comments or bug reports to singular@mathematik.uni-kl.de.

If you want to be informed of new releases, please register as a SINGULAR user by sending an email to singular@mathematik.uni-kl.de with subject line `register` and body containing the following data: your name, email address, organisation, country and platform(s).

For information on how to cite SINGULAR see
https://www.singular.uni-kl.de/index.php/how-to-cite-singular.

You can also support SINGULAR by informing us about your result obtained by using SINGULAR.

Availability

The latest information regarding the status of SINGULAR is always available from https://www.singular.uni-kl.de. The program SINGULAR and the above mentioned parts are available via anonymous ftp through the following addresses:

GMP, libreadline
© Free Software Foundation
https://gmplib.org

NTL © Victor Shoup
http://www.shoup.net/ntl

cdd (C implementation of the Double Description Method of Motzkin et al)
© Komei Fukuda
`http://www-oldurls.inf.ethz.ch/personal/fukudak/cdd_home/`

FLINT
© Bill Hart, Sebastian Pancratz, Fredrik Johansson
`http://www.flintlib.org`

gfanlib
© Anders Jensen
`https://users-math.au.dk/~jensen/software/gfan/gfan.html`

Singular-Factory
© Gert-Martin Greuel/Rüdiger Stobbe/Martin Lee, University of Kaiserslautern:
`https://www.singular.uni-kl.de/ftp/pub/Math/Singular/Factory`

Singular-libfac
© Messollen, University of Saarbrücken:
`ftp://jim.mathematik.uni-kl.de/pub/Math/Singular/Libfac/`

SINGULAR binaries and sources
`ftp://jim.mathematik.uni-kl.de/pub/Math/Singular/` or via a WWW browser from `https://www.singular.uni-kl.de/ftp/pub/Math/Singular/`

Cygwin
`https://www.cygwin.com/`

Xemacs
`https://www.xemacs.org`

Some external programs are optional:

4ti2 (used by sing4ti2.lib, see Section D.4.33 [sing4ti2_lib], page 1381)
`https://4ti2.github.io`

gfan (used by tropical.lib, see Section D.13.6 [tropical_lib], page 2139)
`https://users-math.au.dk/~jensen/software/gfan/gfan.html`

graphviz (used by resgraph.lib, see Section D.5.13 [resgraph_lib], page 1523)
`https://www.graphviz.org/`

normaliz (used by normaliz.lib, see Section D.4.24 [normaliz_lib], page 1209)
© Winfried Bruns and Bogdan Ichim
`https://www.normaliz.uni-osnabrueck.de`

surf (used by surf.lib, see Section D.9.3 [surf_lib], page 1917)
© Stephan Endrass
`http://surf.sf.net`

surfer (used by surf.lib, see Section D.9.3 [surf_lib], page 1917)
`https://imaginary.org/program/surfer`

surfex (used by surfex.lib, see Section D.9.4 [surfex_lib], page 1918)
© Oliver Labs (2001-2008), Stephan Holzer (2004-2005)
`https://github.com/Singular/Singular/tree/spielwiese/Singular/LIB/surfex`

TOPCOM (used by polymake.lib, see Section D.13.4 [polymake_lib], page 2128)
© Jörg Rambau
`http://www.rambau.wm.uni-bayreuth.de/TOPCOM/`

Acknowledgements

The development of Singular is directed and coordinated by Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann.

Current devteams: Abdus Salam School of Mathematical Sciences in Lahore, BTU Cottbus, Center for Advanced Security Research Darmstadt (CASED), FU Berlin, Isfahan University of Technology, Mathematisches Forschunginstitut Oberwolfach, Oklahoma State University, RWTH Aachen, Universidad de Buenos Aires, Université de Versailles Saint-Quentin-en-Yvelines, University of Göttingen, University of Hannover, University of La Laguna and University of Valladolid.

Current Singular developers: Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, Hans Schönemann,

Shawki Al-Rashed, Daniel Andres, Mohamed Barakat, Isabel Bermejo, Muhammad Asan Binyamin, René Birkner, Rocio Blanco, Xenia Bogomolec, Michael Brickenstein, Stanislav Bulygin, Antonio Campillo, Raza Choudery, Alexander Dreyer, Christian Eder, Santiago Encinas, Jose Ignacio Farran, Anne Frühbis-Krüger, Rosa de Frutos, Eva Garcia-Llorente, Ignacio Garcia-Marco, Christian Haase, Amir Hashemi, Fernando Hernando, Bradford Hovinen, Nazeran Idress, Anders Jensen, Lars Kastner, Junaid Alan Khan, Kai Krüger, Santiago Laplagne, Grégoire Lecerf, Martin Lee, Viktor Levandovskyy, Benjamin Lorenz, Christoph Lossen, Thomas Markwig, Hannah Markwig, Irene Marquez, Bernd Martin, Edgar Martinez, Martin Monerjan, Francisco Monserrat, Oleksandr Motsak, Andreas Paffenholz, Maria Jesus Pisabarro, Diego Ruano, Afshan Sadiq, Kristina Schindelar, Mathias Schulze, Frank Seelisch, Andreas Steenpaß, Stefan Steidel, Grischa Studzinski, Katharina Werner and Eva Zerz.

Further contributions to Singular have been made by: Martin Albrecht, Olaf Bachmann, Muhammad Ahsan Banyamin, Thomas Bauer, Thomas Bayer, Markus Becker, J. Boehm, Gergo Gyula Borus, Winfried Bruns, Fernando Hernando Carrillo, Victor Castellanos, Nadine Cremer, Michael Cuntz, Kai Dehmann, Christian Dingler, Marcin Dumnicki, Stephan Endraß, Vladimir Gerdt, Philippe Gimenez, Christian Gorzel, Hubert Grassmann, Jan Hackfeld, Agnes Heydtmann, Dietmar Hillebrand, Tobias Hirsch, Markus Hochstetter, N. Idrees, Manuel Kauers, Simon King, Sebastian Jambor, Oliver Labs, Anen Lakhal, Martin Lamm, Francisco Javier Lobillo, Christoph Mang, Michael Meßollen, Andrea Mindnich, Antonio Montes, Jorge Martin Morales, Thomas Nüßler, Wolfgang Neumann, Markus Perling, Wilfried Pohl, Adrian Popescu, Tetyana Povalyaeva, Carlos Rabelo, Philipp Renner, J.-J.Salazar-Gonzalez, Alfredo Sanchez-Navarro, Ivor Saynisch, Jens Schmidt, Thomas Siebert, Christof Soeger, Silke Spang, William Stein, Rüdiger Stobbe, Henrik Strohmayer, Christian Stussak, Imade Sulandra, Akira Suzuki, Christine Theis, Enrique Tobis, Alberto Vigneron-Tenorio, Moritz Wenk, Eric Westenberger, Tim Wichmann, Oliver Wienand, Denis Yanovich and Oleksandr Yena.

The development of Singular has been supported by the Deutsche Forschungsgemeinschaft (DFG), the Stiftung Rheinland-Pfalz für Innovation, the Volkswagen Stiftung, and the European Union. From 2010 to 2016, it was part of the DFG Priority Project SPP 1489. It is currently funded by the DFG transregional collaborative research centre (SFB-TRR) 195 Symbolic Tools in Mathematics and their Application and the Europeans Unions Horizon 2020 research and innovation programme under grant agreement 676541.

2 Introduction

2.1 Background

Singular is a Computer Algebra system for polynomial computations with emphasis on the special needs of commutative algebra, algebraic geometry, and singularity theory.

Singular's main computational objects are ideals and modules over a large variety of baserings. The baserings are polynomial rings or localizations thereof over a field (e.g., finite fields, the rationals, floats, algebraic extensions, transcendental extensions) or over a limited set of rings, or over quotient rings with respect to an ideal.

Singular features one of the fastest and most general implementations of various algorithms for computing Groebner resp. standard bases. The implementation includes Buchberger's algorithm (if the ordering is a wellordering) and Mora's algorithm (if the ordering is a tangent cone ordering) as special cases. Furthermore, it provides polynomial factorization, resultant, characteristic set and gcd computations, syzygy and free-resolution computations, and many more related functionalities.

Based on an easy-to-use interactive shell and a C-like programming language, Singular's internal functionality is augmented and user-extendible by libraries written in the Singular programming language. A general and efficient implementation of communication links allows Singular to make its functionality available to other programs.

Singular's development started in 1984 with an implementation of Mora's Tangent Cone algorithm in Modula-2 on an Atari computer (K.P. Neuendorf, G. Pfister, H. Schönemann; Humboldt-Universität zu Berlin). The need for a new system arose from the investigation of mathematical problems coming from singularity theory which none of the existing systems was able to handle.

In the early 1990s Singular's "home-town" moved to Kaiserslautern, a general standard basis algorithm was implemented in C and Singular was ported to Unix, MS-DOS, Windows NT, and MacOS.

Continuous extensions (like polynomial factorization, gcd computations, links) and refinements led in 1997 to the release of Singular version 1.0 and in 1998 to the release of version 1.2 (with a much faster standard and Groebner bases computation based on Hilbert series and on an improved implementation of the core algorithms, libraries for primary decomposition, ring normalization, etc.)

For the highlights of the new Singular version 4.3.2, see Section 8.1 [News and changes], page 2577.

2.2 How to use this manual

For the impatient user

In Section 2.3 [Getting started], page 6, some simple examples explain how to use Singular in a step-by-step manner.

Appendix A [Examples], page 698 should come next for real learning-by-doing or to quickly solve some given mathematical problem without dwelling too deeply into Singular. This chapter contains a lot of real-life examples and detailed instructions and explanations on how to solve mathematical problems using Singular.

For the systematic user

In Chapter 3 [General concepts], page 15, all basic concepts which are important to use and to understand Singular are developed. But even for users preferring the systematic approach it will be helpful to take a look at the examples in Section 2.3 [Getting started], page 6, every now and then. The topics in the chapter are organized more or less in the natural order in which the novice user is expected to have to deal with them.

- In Section 3.1 [Interactive use], page 15, and its subsections there are some words on entering and exiting Singular, followed by a number of other aspects concerning the interactive user-interface.
- To do anything more than trivial integer computations, one needs to define a basering in Singular. This is explained in detail in Section 3.3 [Rings and orderings], page 30.
- An overview of the algorithms implemented in the kernel of Singular is given in Section 3.4 [Implemented algorithms], page 37.
- In Section 3.5 [The SINGULAR language], page 41, language specific concepts are introduced, such as the notions of names and objects, data types and conversion between them, etc.
- In Section 3.6 [Input and output], page 48, Singular's mechanisms to store and retrieve data are discussed.
- The more complex concepts of procedures and libraries as well as tools for debugging them are considered in the following sections: Section 3.7 [Procedures], page 50, Section 3.8 [Libraries], page 55, and Section 3.9 [Debugging tools], page 68.

Chapter 4 [Data types], page 73, is a complete treatment of Singular's data types in alphabetical order, where each section corresponds to one data type. For each data type, its purpose is explained, the syntax of its declaration is given, related operations and functions are listed, and one or more examples illustrate its usage.

Chapter 5 [Functions and system variables], page 156, is an alphabetically ordered reference list of all of Singular's functions, control structures, and system variables. Each entry includes a description of the syntax and semantics of the item being explained as well as one or more examples on how to use it.

Miscellaneous

Chapter 6 [Tricks and pitfalls], page 309, is a loose collection of limitations and features which may be unexpected by those who expect the Singular language to be an exact copy of the C programming language or of some other Computer Algebra system's language. Additionally, some mathematical hints are collected there.

Appendix C [Mathematical background], page 774, introduces some of the mathematical notions and definitions used throughout this manual. For example, if in doubt what exactly a "negative degree reverse lexicographical ordering" is in Singular, one should refer to this chapter.

Appendix D [SINGULAR libraries], page 793, lists the libraries which come with Singular, and all functions contained in them.

Typographical conventions

Throughout this manual, the following typographical conventions are adopted:

- text in `typewriter` denotes Singular input and output as well as reserved names:

  The basering can, e.g., be set using the command `setring`.

- the arrow ↦ denotes Singular output:

```
poly p=x+y+z;
p*p;
↦ x2+2xy+y2+2xz+2yz+z2
```

- square brackets are used to denote parts of syntax descriptions which are optional:

  [optional_text] required_text
- keys are denoted using typewriter, for example:

  `N` (press the key `N` to get to the next node in help mode)

  `RETURN` (press `RETURN` to finish an input line)

  `CTRL-P` (press the control key together with the key `P` to get the previous input line)

2.3 Getting started

Singular is a special purpose system for polynomial computations. Hence, most of the powerful computations in Singular require the prior definition of a ring. Most important rings are polynomial rings over a field, localizations thereof, or quotient rings of such rings modulo an ideal. However, some simple computations with integers (machine integers of limited size) and manipulations of strings can be carried out without the prior definition of a ring.

2.3.1 First steps

Once Singular is started, it awaits an input after the prompt `>`. Every statement has to be terminated by `;` .

```
37+5;
↦ 42
```

All objects have a type, e.g., integer variables are defined by the word `int`. An assignment is made using the symbol `=` .

```
int k = 2;
```

Test for equality resp. inequality is done using `==` resp. `!=` (or `<>`), where `0` represents the boolean value FALSE, and any other value represents TRUE.

```
k == 2;
↦ 1
k != 2;
↦ 0
```

The value of an object is displayed by simply typing its name.

```
k;
↦ 2
```

On the other hand, the output is suppressed if an assignment is made.

```
int j;
j = k+1;
```

The last displayed (!) result can be retrieved via the special symbol `_` .

```
2*_;   // the value from k displayed above
↦ 4
```

Text starting with `//` denotes a comment and is ignored in calculations, as seen in the previous example. Furthermore Singular maintains a history of the previous lines of input, which may be accessed by `CTRL-P` (previous) and `CTRL-N` (next) or the arrows on the keyboard.

The whole manual is available online by typing the command `help;` . Documentation on single topics, e.g., on `intmat`, which defines a matrix of integers, is obtained by

```
help intmat;
```

This will display the text of Section 4.8 [intmat], page 90, in the printed manual.

Next, we define a 3 × 3 matrix of integers and initialize it with some values, row by row from left to right:

```
intmat m[3][3] = 1,2,3,4,5,6,7,8,9;
m;
```

A single matrix entry may be selected and changed using square brackets [and].

```
m[1,2]=0;
m;
↦ 1,0,3,
↦ 4,5,6,
↦ 7,8,9
```

To calculate the trace of this matrix, we use a `for` loop. The curly brackets { and } denote the beginning resp. end of a block. If you define a variable without giving an initial value, as the variable `tr` in the example below, Singular assigns a default value for the specific type. In this case, the default value for integers is `0`. Note that the integer variable `j` has already been defined above.

```
int tr;
for ( j=1; j <= 3; j++ ) { tr=tr + m[j,j]; }
tr;
↦ 15
```

Variables of type string can also be defined and used without having an active ring. Strings are delimited by `"` (double quotes). They may be used to comment the output of a computation or to give it a nice format. If a string contains valid Singular commands, it can be executed using the function `execute`. The result is the same as if the commands would have been written on the command line. This feature is especially useful to define new rings inside procedures.

```
"example for strings:";
↦ example for strings:
string s="The element of m ";
s = s + "at position [2,3] is:";  // concatenation of strings by +
s , m[2,3] , ".";
↦ The element of m at position [2,3] is: 6 .
s="m[2,1]=0; m;";
execute(s);
↦ 1,0,3,
↦ 0,5,6,
↦ 7,8,9
```

This example shows that expressions can be separated by `,` (comma) giving a list of expressions. Singular evaluates each expression in this list and prints all results separated by spaces.

2.3.2 Rings and standard bases

In order to compute with objects such as ideals, matrices, modules, and polynomial vectors, a ring has to be defined first.

```
ring r = 0,(x,y,z),dp;
```

The definition of a ring consists of three parts: the first part determines the ground field, the second part determines the names of the ring variables, and the third part determines the monomial ordering to be used. Thus, the above example declares a polynomial ring called `r` with a ground field of characteristic 0 (i.e., the rational numbers) and ring variables called `x`, `y`, and `z`. The `dp` at the end determines that the degree reverse lexicographical ordering will be used.

Other ring declarations:

`ring r1=32003,(x,y,z),dp;`
: characteristic 32003, variables `x`, `y`, and `z` and ordering `dp`.

`ring r2=32003,(a,b,c,d),lp;`
: characteristic 32003, variable names `a`, `b`, `c`, `d` and lexicographical ordering.

`ring r3=7,(x(1..10)),ds;`
: characteristic 7, variable names `x(1)`,...,`x(10)`, negative degree reverse lexicographical ordering (`ds`).

`ring r4=(0,a),(mu,nu),lp;`
: transcendental extension of Q by a , variable names `mu` and `nu`, lexicographical ordering.

`ring r5=real,(a,b),lp;`
: floating point numbers (single machine precision), variable names `a` and `b`.

`ring r6=(real,50),(a,b),lp;`
: floating point numbers with precision extended to 50 digits, variable names `a` and `b`.

`ring r7=(complex,50,i),(a,b),lp;`
: complex floating point numbers with precision extended to 50 digits and imaginary unit `i`, variable names `a` and `b`.

`ring r8=integer,(a,b),lp;`
: the ring of integers (see Section 3.3.4 [Coefficient rings], page 37), variable names `a` and `b`.

`ring r9=(integer, 60),(a,b),lp;`
: the ring of integers modulo 60 (see Section 3.3.4 [Coefficient rings], page 37), variable names `a` and `b`.

`ring r10=(integer, 2, 10),(a,b),lp;`
: the ring of integers modulo 2^10 (see Section 3.3.4 [Coefficient rings], page 37), variable names `a` and `b`.

Typing the name of a ring prints its definition. The example below shows that the default ring in SINGULAR is $Z/32003[x,y,z]$ with degree reverse lexicographical ordering:

```
ring r11;
r11;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

Defining a ring makes this ring the current active basering, so each ring definition above switches to a new basering. The concept of rings in SINGULAR is discussed in detail in Section 3.3 [Rings and orderings], page 30.

The basering is now `r11`. Since we want to calculate in the ring `r`, which we defined first, we need to switch back to it. This can be done using the function `setring`:

```
setring r;
```

Once a ring is active, we can define polynomials. A monomial, say x^3, may be entered in two ways: either using the power operator `^`, writing `x^3`, or in short-hand notation without operator, writing x3. Note that the short-hand notation is forbidden if a name of the ring variable(s) consists of more than one character(see Section 6.4 [Miscellaneous oddities], page 313 for details). Note, that SINGULAR always expands brackets and automatically sorts the terms with respect to the monomial ordering of the basering.

```
poly f =  x3+y3+(x-y)*x2y2+z2;
f;
↦ x3y2-x2y3+x3+y3+z2
```

The command size retrieves in general the number of entries in an object. In particular, for polynomials, size returns the number of monomials.

```
size(f);
↦ 5
```

A natural question is to ask if a point, e.g., (x,y,z)=(1,2,0), lies on the variety defined by the polynomials f and g. For this we define an ideal generated by both polynomials, substitute the coordinates of the point for the ring variables, and check if the result is zero:

```
poly g =  f^2 *(2x-y);
ideal I = f,g;
ideal J = subst(I,var(1),1);
J = subst(J,var(2),2);
J = subst(J,var(3),0);
J;
↦ J[1]=5
↦ J[2]=0
```

Since the result is not zero, the point (1,2,0) does not lie on the variety V(f,g).

Another question is to decide whether some function vanishes on a variety, or in algebraic terms, if a polynomial is contained in a given ideal. For this we calculate a standard basis using the command groebner and afterwards reduce the polynomial with respect to this standard basis.

```
ideal sI = groebner(f);
reduce(g,sI);
↦ 0
```

As the result is 0 the polynomial g belongs to the ideal defined by f.

The function groebner, like many other functions in SINGULAR, prints a protocol during calculations, if desired. The command option(prot); enables protocolling whereas option(noprot); turns it off. Section 5.1.111 [option], page 234, explains the meaning of the different symbols printed during calculations.

The command kbase calculates a basis of the polynomial ring modulo an ideal, if the quotient ring is finite dimensional. As an example we calculate the Milnor number of a hypersurface singularity in the global and local case. This is the vector space dimension of the polynomial ring modulo the Jacobian ideal in the global case resp. of the power series ring modulo the Jacobian ideal in the local case. See Section A.4.2 [Critical points], page 736, for a detailed explanation.

The Jacobian ideal is obtained with the command jacob.

```
ideal J = jacob(f);
↦ // ** redefining J **
J;
↦ J[1]=3x2y2-2xy3+3x2
↦ J[2]=2x3y-3x2y2+3y2
↦ J[3]=2z
```

SINGULAR prints the line // ** redefining J **. This indicates that we had previously defined a variable with name J of type ideal (see above).

To obtain a representing set of the quotient vector space we first calculate a standard basis, and then apply the function `kbase` to this standard basis.

```
J = groebner(J);
ideal K = kbase(J);
K;
↦ K[1]=y4
↦ K[2]=xy3
↦ K[3]=y3
↦ K[4]=xy2
↦ K[5]=y2
↦ K[6]=x2y
↦ K[7]=xy
↦ K[8]=y
↦ K[9]=x3
↦ K[10]=x2
↦ K[11]=x
↦ K[12]=1
```

Then

```
size(K);
↦ 12
```

gives the desired vector space dimension $K[x, y, z]/\text{jacob}(f)$. As in Singular the functions may take the input directly from earlier calculations, the whole sequence of commands may be written in one single statement.

```
size(kbase(groebner(jacob(f))));
↦ 12
```

When we are not interested in a basis of the quotient vector space, but only in the resulting dimension we may even use the command `vdim` and write:

```
vdim(groebner(jacob(f)));
↦ 12
```

2.3.3 Procedures and libraries

Singular offers a comfortable programming language, with a syntax close to C. So it is possible to define procedures which bind a sequence of several commands in a new one. Procedures are defined using the keyword `proc` followed by a name and an optional parameter list with specified types. Finally, a procedure may return a value using the command `return`.

We may e.g. define the following procedure called `Milnor`: (Here the parameter list is `(poly h)` meaning that `Milnor` must be called with one argument which can be assigned to the type `poly` and is referred to by the name `h`.)

```
proc Milnor (poly h)
{
  return(vdim(groebner(jacob(h))));
}
```

Note: if you have entered the first line of the procedure and pressed `RETURN`, Singular prints the prompt `.` (dot) instead of the usual prompt `>` . This shows that the input is incomplete and Singular expects more lines. After typing the closing curly bracket, Singular prints the usual prompt indicating that the input is now complete.

Then we can call the procedure:

```
Milnor(f);
↦ 12
```

Note that the result may depend on the basering as we will see in the next chapter.

The distribution of Singular contains several libraries, each of which is a collection of useful procedures based on the kernel commands, which extend the functionality of Singular. The command `listvar(package);` list all currently loaded libraries. The command `LIB "all.lib";` loads all libraries.

One of these libraries is `sing.lib` which already contains a procedure called `milnor` to calculate the Milnor number not only for hypersurfaces but more generally for complete intersection singularities.

Libraries are loaded using the command `LIB`. Some additional information during the process of loading is displayed on the screen, which we omit here.

```
LIB "sing.lib";
```

As all input in Singular is case sensitive, there is no conflict with the previously defined procedure `Milnor`, but the result is the same.

```
milnor(f);
↦ 12
```

The procedures in a library have a help part which is displayed by typing

```
help milnor;
```

as well as some examples, which are executed by

```
example milnor;
```

Likewise, the library itself has a help part, to show a list of all the functions available for the user which are contained in the library.

```
help sing.lib;
```

The output of the help commands is omitted here.

2.3.4 Change of rings

To calculate the local Milnor number we have to do the calculation with the same commands in a ring with local ordering. We can define the localization of the polynomial ring at the origin (see Appendix B [Polynomial data], page 767, and Appendix C [Mathematical background], page 774).

```
ring rl = 0,(x,y,z),ds;
```

The ordering directly affects the standard basis which will be calculated. Fetching the polynomial defined in the ring `r` into this new ring, helps us to avoid retyping previous input.

```
poly f = fetch(r,f);
f;
↦ z2+x3+y3+x3y2-x2y3
```

Instead of `fetch` we can use the function `imap` which is more general but less efficient. The most general way to fetch data from one ring to another is to use maps, this will be explained in Section 4.12 [map], page 106.

In this ring the terms are ordered by increasing exponents. The local Milnor number is now

```
Milnor(f);
↦ 4
```

This shows that `f` has outside the origin in affine 3-space singularities with local Milnor number adding up to $12 - 4 = 8$. Using global and local orderings as above is a convenient way to check whether a variety has singularities outside the origin.

The command `jacob` applied twice gives the Hessian of `f`, in our example a 3x3 - matrix.

```
matrix H = jacob(jacob(f));
H;
↦ H[1,1]=6x+6xy2-2y3
↦ H[1,2]=6x2y-6xy2
↦ H[1,3]=0
↦ H[2,1]=6x2y-6xy2
↦ H[2,2]=6y+2x3-6x2y
↦ H[2,3]=0
↦ H[3,1]=0
↦ H[3,2]=0
↦ H[3,3]=2
```

The `print` command displays the matrix in a nicer format.

```
print(H);
↦ 6x+6xy2-2y3,6x2y-6xy2,  0,
↦ 6x2y-6xy2,  6y+2x3-6x2y,0,
↦ 0,          0,          2
```

We may calculate the determinant and (the ideal generated by all) minors of a given size.

```
det(H);
↦ 72xy+24x4-72x3y+72xy3-24y4-48x4y2+64x3y3-48x2y4
minor(H,1);  // the 1x1 - minors
↦ _[1]=2
↦ _[2]=6y+2x3-6x2y
↦ _[3]=6x2y-6xy2
↦ _[4]=6x2y-6xy2
↦ _[5]=6x+6xy2-2y3
```

The algorithm of the standard basis computation may be affected by the command `option`. For example, a reduced standard basis of the ideal generated by the 1×1-minors of H is obtained in the following way:

```
option(redSB);
groebner(minor(H,1));
↦ _[1]=1
```

This shows that 1 is contained in the ideal of the 1×1-minors, hence the corresponding variety is empty.

2.3.5 Modules and their annihilator

Now we shall give three more advanced examples.

Singular is able to handle modules over all the rings, which can be defined as a basering. A free module of rank `n` is defined as follows:

```
ring rr;
int n = 4;
freemodule(4);
↦ _[1]=gen(1)
↦ _[2]=gen(2)
↦ _[3]=gen(3)
↦ _[4]=gen(4)
typeof(_);
↦ module
print(freemodule(4));
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,0,1,0,
↦ 0,0,0,1
```

To define a module, we provide a list of vectors generating a submodule of a free module. Then this set of vectors may be identified with the columns of a matrix. For that reason in Singular matrices and modules may be interchanged. However, the representation is different (modules may be considered as sparse matrices).

```
ring r =0,(x,y,z),dp;
module MD = [x,0,x],[y,z,-y],[0,z,-2y];
matrix MM = MD;
print(MM);
↦ x,y,0,
↦ 0,z,z,
↦ x,-y,-2y
```

However the submodule MD may also be considered as the module of relations of the factor module r^3/MD. In this way, Singular can treat arbitrary finitely generated modules over the basering (see Section B.1 [Representation of mathematical objects], page 767).

In order to get the module of relations of MD , we use the command `syz`.

```
syz(MD);
↦ _[1]=x*gen(3)-x*gen(2)+y*gen(1)
```

We want to calculate, as an application, the annihilator of a given module. Let $M = r^3/U$, where U is our defining module of relations for the module M.

```
module U = [z3,xy2,x3],[yz2,1,xy5z+z3],[y2z,0,x3],[xyz+x2,y2,0],[xyz,x2y,1];
```

Then, by definition, the annihilator of M is the ideal $\text{ann}(M) = \{a \mid aM = 0\}$ which is, by definition of M, the same as $\{a \mid ar^3 \in U\}$. Hence we have to calculate the quotient $U\colon r^3$. The rank of the free module is determined by the choice of U and is the number of rows of the corresponding matrix. This may be retrieved by the function `nrows`. All we have to do now is the following:

```
quotient(U,freemodule(nrows(U)));
```

The result is too big to be shown here.

2.3.6 Resolution

There are several commands in Singular for computing free resolutions. The most general command is `res(... ,n)` which determines heuristically what method to use for the given problem. It computes the free resolution up to the length n , where $n = 0$ corresponds to the full resolution.

Here we use the possibility to inspect the calculation process using the option `prot`.

```
ring R;       // the default ring in char 32003
R;
↦ //   characteristic : 32003
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
ideal I = x4+x3y+x2yz,x2y2+xy2z+y2z2,x2z2+2xz3,2x2z2+xyz2;
option(prot);
resolution rs = res(I,0);
↦ using lres
↦ 4(m0)4(m1).5(m1)g.g6(m1)...6(m2)..
```

Disable this protocol with

```
option(noprot);
```

When we enter the name of the calculated resolution, we get a pictorial description of the minimized resolution where the exponents denote the rank of the free modules. Note that the calculated resolution itself may not yet be minimal.

```
rs;
↦  1      4      5      2      0
↦ R  <-- R  <-- R  <-- R  <-- R
↦ 
↦ 0      1      2      3      4
print(betti(rs),"betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     -     -     -
↦     2:     -     -     -     -
↦     3:     -     4     1     -
↦     4:     -     -     1     -
↦     5:     -     -     3     2
↦ ------------------------------
↦ total:     1     4     5     2
```

In order to minimize the resolution, that is to calculate the maps of the minimal free resolution, we use the command `minres`:

```
rs=minres(rs);
```

A single module in this resolution is obtained (as usual) with the brackets `[` and `]`. The `print` command can be used to display a module in a more readable format:

```
print(rs[3]);
↦ z3,   -xyz-y2z-4xz2+16z3,
↦ 0,    -y2,
↦ -y+4z,48z,
↦ x+2z, 48z,
↦ 0,    x+y-z
```

In this case, the output is to be interpreted as follows: the 3rd syzygy module of R/I, `rs[3]`, is the rank-2-submodule of R^5 generated by the vectors $(z^3, 0, -y+4z, x+2z, 0)$ and $(-xyz-y^2z-4xz^2+16z^3, -y^2, 48z, 48z, x+y-z)$.

3 General concepts

3.1 Interactive use

In this section, aspects of interactive use are discussed. This includes how to enter and exit SINGULAR, how to interpret its prompt, how to get online help, and so on.

There are a few important notes which one should not forget:

- every command has to be terminated by a ; (semicolon) followed by a ⟨RETURN⟩
- the online help is accessible by means of the help function

3.1.1 How to enter and exit

SINGULAR can either be run in an ASCII-terminal or within Emacs.

To start SINGULAR in its ASCII-terminal user interface, enter `Singular` at the system prompt. The SINGULAR banner appears which, among other data, reports the version and the compilation date.

To start SINGULAR in its Emacs user interface, either enter `ESingular` at the system prompt, or type `M-x singular` within a running Emacs (provided you have loaded the file `singular.el` in your running Emacs, see Section 3.2.2 [Running SINGULAR under Emacs], page 25 for details).

Generally, we recommend to use SINGULAR in its Emacs interface, since this offers many more features and is more convenient to use than the ASCII-terminal interface (see Section 3.2 [Emacs user interface], page 22).

To exit SINGULAR type `quit;`, `exit;` or `$` (or, when running within Emacs preferably type `C-c $`).

SINGULAR and `ESingular` may also be started with command line options and with filenames as arguments. More generally, the startup syntax is

```
Singular   [options] [file1 [file2 ...]]
ESingular  [options] [file1 [file2 ...]]
```

See Section 3.1.6 [Command line options], page 19, Section 3.1.7 [Startup sequence], page 22, Section 3.2.2 [Running SINGULAR under Emacs], page 25.

3.1.2 The SINGULAR prompt

The SINGULAR prompt `>` (larger than) asks the user for input of commands. The “continuation” prompt `.` (period) asks the user for input of missing parts of a command (e.g. the semicolon at the end of every command).

SINGULAR does not interpret the semicolon as the end of a command if it occurs inside a string. Also, SINGULAR waits for blocks (sequences of commands enclosed in curly brackets) to be closed before prompting with `>` for more commands. Thus, if SINGULAR does not respond with its regular prompt after typing a semicolon it may wait for a `"` or a `}` first.

Additional semicolons will not harm SINGULAR since they are interpreted as empty statements.

3.1.3 The online help system

The online help system is invoked by the `help` command. `?` may be used as a synonym for `help`. Simply typing `help;` displays the “top” of the help system (i.e., the title page of the SINGULAR manual) which offers a short table of contents. Typing `help` topic`;` shows the available documentation on the respective topic. Here, topic may be either a function name or, more generally, any index entry of the Singular manual. Furthermore, topic may contain wildcard characters. See Section 5.1.54 [help], page 193, for more information.

Online help information can be displayed in various help browsers. The following table lists a summary of the browsers which are always present. Usually, external browsers are much more convenient: A complete, customizable list can be found in the file `LIB/help.cnf`.

Browser	**Platform**	**Description**
html	Windows	displays a html version of the manual in your default html browser
builtin	all	simply outputs the help information in plain ASCII format
emacs	Unix, Windows	when running Singular within (X)emacs, displays help inside the (X)emacs info buffer.
dummy	all	displays an error message due to the non-availability of a help browser

External browsers depend on your system and the contents of `LIB/help.cnf`, the default includes:
`htmlview` (displays HTML help pages via `htlmview`),
`mac` (displays HTML help pages via `open`),
`mac-net` (displays HTML help pages via `open`),
`mozilla` (displays HTML help pages via `mozilla`),
`firefox` (displays HTML help pages via `firefox`),
`konqueror` (displays HTML help pages via `konqueror`),
`galeon` (displays HTML help pages via `galeon`),
`netscape` (displays HTML help pages via `netscape`),
`safari` (displays HTML help pages on MacOsX via `safari`),
`tkinfo` (displays INFO help pages via `tkinfo`),
`xinfo` (displays INFO help pages via `info`),
`info` (displays INFO help pages via `info`),
`lynx` (displays HTML help pages via `lynx`).

The browser which is used to display the help information, can be either set at startup time with the command line option (see Section 3.1.6 [Command line options], page 19)

```
--browser=<browser>
```

or with the Singular command (see Section 5.1.155 [system], page 275)

```
system("--browser", "<browser>");
```

The Singular command

```
system("browsers");
```

lists all available browsers and the command

```
system("--browser");
```

returns the currently used browser.

If no browser is explicitly set by the user, then the first available browser (w.r.t. the order of the browsers in the file `LIB/help.cnf`) is chosen.

The `.singularrc` (see Section 3.1.7 [Startup sequence], page 22) file is a good place to set your default browser. Recall that if a file `$HOME/.singularrc` exists on your system, then the content of this file is executed before the first user input. Hence, putting

```
if (! system("--emacs"))
{
  // only set help browser if not running within emacs
  system("--browser", "info");
}
// if help browser is later on set to a web browser,
// allow it to fetch HTML pages from the net
system("--allow-net", 1);
```

in your file `$HOME/.singularrc` sets your default browser to `info`, unless Singular is run within emacs (in which case the default browser is automatically set to `emacs`).

Obviously, certain external files and programs are required for the Singular help system to work correctly. If something is not available or goes wrong, here are some tips for troubleshooting the help system:

- Under Unix, the environment variable `DISPLAY` has to be set for all X11 browsers to work.
- The help browsers are only available if the respective programs are installed on your system (for `xinfo`, the programs `xterm` and `info` are necessary). You can explicitly specify which program to use, by changing the entry in `LIB/help.cnf`
- If the help browser cannot find the local html pages of the Singular manual (which it will look for at `$RootDir/html` – see Section 3.8.11 [Loading a library], page 66 for more info on `$RootDir`) *and* the (command-line) option `--allow-net` has *explicitly* been set (see Section 3.1.6 [Command line options], page 19 and Section 5.1.155 [system], page 275 for more info on setting values of command-line options), then it dispatches the html pages from `https://www.singular.uni-kl.de/Manual`. (Note that the non-local net-access of HTML pages is disabled, by default.)
  An alternative location of a local directory where the html pages reside can be specified by setting the environment variable `SINGULAR_HTML_DIR`.
- The `info` based help browsers `tkinfo`, `xinfo`, `info`, and `builtin` need the (info) file `singular.info` which will be looked for at `$RootDir/info/singular.info` (see Section 3.8.11 [Loading a library], page 66 for more info on `$RootDir`). An alternative location of the info file of the manual can be specified by setting the environment variable `SINGULAR_INFO_FILE`.

Section 3.1.6 [Command line options], page 19

Info help browsers

The help browsers `tkinfo`, `xinfo` and `info` (so-called info help browsers) are based on the `info` program from the GNU `texinfo` package. See section "Getting started" in *The Info Manual*, for more information.

For info help browsers, the online manual is decomposed into "nodes" of information, closely related to the division of the printed manual into sections and subsections. A node contains text describing a specific topic at a specific level of detail. The top line of a node is its "header". The node's header tells the name of the current node (`Node:`), the name of the next node (`Next:`), the name of the previous node (`Prev:`), and the name of the upper node (`Up:`).

To move within info, type commands consisting of single characters. Do not type `RETURN`. Do not use cursor keys, either. Using some of the cursor keys by accident might pop to some totally different node. Type `l` to return to the original node. Some of the `info` commands read input from the command line at the bottom. The `TAB` key may be used to complete partially entered input.

The most important commands are:

`q` leaves the online help system

`n` goes to the next node

`p` goes to the previous node

`u` goes to the upper node

`m` picks a menu item specified by name

`f` follows a cross reference

`l` goes to the previously visited node

`b` goes to the beginning of the current node

`e` goes to the end of the current node

`SPACE` scrolls forward a page

`DEL` scrolls backward a page

`h` invokes info tutorial (use `l` to return to the manual or `CTRL-X 0` to remove extra window)

`CTRL-H` shows a short overview over the online help system (use `l` to return to the manual or `CTRL-X 0` to remove extra window)

`s` searches through the manual for a specific string, and selects the node in which the next occurrence is found

`1`, ..., `9` picks i-th subtopic from a menu

3.1.4 Interrupting SINGULAR

On Unix-like operating systems and on Windows NT, typing `CTRL-C` (or, alternatively `C-c C-c`, when running within Emacs), interrupts Singular. Singular prints the current command and the current line and prompts for further action. The following choices are available:

`a` returns to the top level after finishing the current (kernel) command. Notice that commands of the Singular kernel (like `std`) cannot be aborted, i.e. (`a`)bort only happens whenever the interpreter is active.

`c` continues

`q` quits Singular

3.1.5 Editing input

The following keys can be used for editing the input and retrieving previous input lines:

`TAB` provides command line completion for function names and file names

`CTRL-B` moves cursor to the left

`CTRL-F` moves cursor to the right

`CTRL-A` moves cursor to the beginning of the line

`CTRL-E` moves cursor to the end of the line

`CTRL-D` deletes the character under the cursor
Warning: on an empty line, `CTRL-D` is interpreted as the `EOF` character which immediately terminates Singular.

`BACKSPACE`
`DELETE`
`CTRL-H` deletes the character before the cursor

`CTRL-K` kills from cursor to the end of the line

`CTRL-U` kills from cursor to the beginning of the line

`CTRL-N` saves the current line to history and gives the next line

`CTRL-P` saves the current line to history and gives the previous line

`RETURN` saves the current line to the history and sends it to the SINGULAR parser for interpretation

When run under a Unix-like operating system and in its ASCII-terminal user interface, SINGULAR tries to dynamically link at runtime with the GNU Readline library. See section "Command Line Editing" in *The GNU Readline Library Manual*, for more information. If a shared version of this library can be found on your machine, then additional command-line editing features like history completion are available. In particular, if SINGULAR is able to load that library the input history is stored across sessions using the file given in the environment variable `SINGULARHIST`. If `SINGULARHIST` is not set `.singularhistory` is used. Otherwise, i.e., if the environment variable `SINGULARHIST` is set to the empty string, the history of the last inputs is only available for previous commands of the current session.

3.1.6 Command line options

The startup syntax is

```
Singular  [options] [file1 [file2 ...]]
ESingular  [options] [file1 [file2 ...]]
```

Options can be given in both their long and short format. The following options control the general behaviour of SINGULAR:

`-d`, `--sdb` Enable the use of the source code debugger. See Section 3.9.3 [Source code debugger], page 69.

`-e`, `--echo[=VAL]`
Set value of variable `echo` to `VAL` (integer in the range 0, . . . , 9). Without an argument, `echo` is set to 1, which echoes all input coming from a file. By default, the value of `echo` is 0. See Section 5.3.2 [echo], page 303.

`-h`, `--help`
Print a one-line description of each command line option and exit.

`--allow-net`
Allow the help browsers based on a web browser to fetch HTML manual pages over the net from the WWW home-site of SINGULAR. See Section 3.1.3 [The online help system], page 15, for more info.

`--browser="VAL"`
Use `VAL` as browser for the SINGULAR online manual.
`VAL` may be one of the browsers mentioned in `LIB/help.cnf`, for example `html` (Windows only), `mozilla`, `firefox`, `konqueror`, `galeon`, `netscape`, `safari` (OsX only), `xinfo`, `tkinfo`, `info`, `builtin`, or `emacs`. Depending on your platform and local installation, only some browsers might be available. The default browser is `html` for Windows and one based on a web browser for Unix platforms. See Section 3.1.3 [The online help system], page 15, for more info.

`--no-rc` Do not execute the `.singularrc` file on start-up. By default, this file is executed on start-up. See Section 3.1.7 [Startup sequence], page 22.

`--no-stdlib`
Do not load the library `standard.lib` on start-up. By default, this library is loaded on start-up. See Section 3.1.7 [Startup sequence], page 22.

`--no-warn`
Do not display warning messages.

`--no-out` Suppress display of all output.

`--no-shell`
Runs Singular in restricted mode to disallow shell escape commands. Objects of type link will also be unable to use.

`-t`, `--no-tty`
Do not redefine the characteristics of the terminal. This option should be used for batch processes.

`-q`, `--quiet`
Do not print the start-up banner and messages when loading libraries. Furthermore, redirect `stderr` (all error messages) to `stdout` (normal output channel). This option should be used if Singular's output is redirected to a file.

`-v` Print extended information about the version and configuration of Singular (used optional parts, compilation date, start of random generator etc.). This information should be included if a user reports an error to the authors.
It also list all the used directories/files (see Section 8.5 [Used environment variables], page 2589).

The following command line options allow manipulations of the timer and the pseudo random generator and enable the passing of commands and strings to Singular:

`-c`, `--execute=STRING`
Execute `STRING` as (a sequence of) Singular commands on start-up after the `.singularrc` file is executed, but prior to executing the files given on the command line. E.g., `Singular -c "help all.lib; quit;"` shows the help for the library `all.lib` and exits.

`-u`, `--user-option=STRING`
Returns `STRING` on `system("--user-option")`. This is useful for passing arbitrary arguments from the command line to the Singular interpreter. E.g.,
`Singular -u "xxx.dump" -c 'getdump(system("--user-option"))'` reads the file `xxx.dump` at start-up and allows the user to start working with all the objects defined in a previous session.

`-r`, `--random=SEED`
Seed (i.e., set the initial value of) the pseudo random generator with integer `SEED`. If this option is not given, then the random generator is seeded with a time-based `SEED` (the number of seconds since January, 1, 1970, on Unix-like operating systems, to be precise).

`--min-time=SECS`
If the `timer` (see Section 5.3.8 [timer], page 305), resp. `rtimer` (see Section 5.3.10 [rtimer], page 308) , variable is set, report only times larger than `SECS` seconds (`SECS` needs to be a floating point number greater than 0). By default, this value is set to 0.5

(i.e., half a second). E.g., the option `--min-time=0.01` forces SINGULAR to report all times larger than 1/100 of a second.

`--ticks-per-sec=TICKS`
: Set unit of timer to `TICKS` ticks per second (i.e., the value reported by the `timer` and `rtimer` variable divided by `TICKS` gives the time in seconds). By default, this value is 1.

`--cpus=CPUs`
: set the maximal number of CPUs to use.

`--cntrlc=C`
: set the default answer for interrupt signals to C which should be a for abort, c for continue or q for quit.

The next three options are of interest for the use with ssi links:

`-b`, `--batch`
: Run in batch mode. Opens a TCP/IP connection with host specified by `--MPhost` at the port specified by `--MPport`. Input is read from and output is written to this connection in the format given by `--link`. See Section 4.10.5 [Ssi links], page 99.

`--MPport=PORT`
: Use `PORT` as default port number for connections (whenever not further specified). This option is mandatory when the `--batch` option is given. See Section 4.10.5 [Ssi links], page 99.

`--MPhost=HOST`
: Use `HOST` as default host for connections (whenever not further specified). This option is mandatory when the `--batch` option is given. See Section 4.10.5 [Ssi links], page 99.

Finally, the following options are only available when running `ESingular` (see Section 3.2.2 [Running SINGULAR under Emacs], page 25 for details).

`--emacs=EMACS`
: Use `EMACS` as Emacs program to run the SINGULAR Emacs interface, where `EMACS` may e.g. be emacs or xemacs.

`--emacs-dir=DIR`
: Set the singular-emacs-home-directory, which is the directory where singular.el can be found, to `DIR`.

`--emacs-load=FILE`
: Load `FILE` on Emacs start-up, instead of the default load file.

`--singular=PROG`
: Start `PROG` as SINGULAR program within Emacs

The value of options given to SINGULAR (resp. their default values, if an option was not given), can be checked with the command `system("--long_option_name")`. See Section 5.1.155 [system], page 275.

```
  system("--quiet");    // if ‘‘quiet’’ 1, otherwise 0
↦ 1
  system("--min-time"); // minimal reported time
↦ 0.5
  system("--random");   // seed of the random generator
↦ 12345678
```

Furthermore, the value of options (e.g., `--browser`) can be re-defined while Singular is running using the command `system("--long_option_name_string ",expression)`. See Section 5.1.155 [system], page 275.

```
system("--browser", "builtin");  // sets browser to 'builtin'
system("--ticks-per-sec", 100);  // sets timer resolution to 100
```

3.1.7 Startup sequence

On start-up, Singular

1. loads the library `standard.lib` (provided the `--no-stdlib` option was not given),
2. searches the current directory and then the home directory of the user, and then all directories contained in the library `SearchPath` (see Section 3.8.11 [Loading a library], page 66 for more info on `SearchPath`) for a file named `.singularrc` and executes it, if found (provided the `--no-rc` option was not given),
3. executes the string specified with the `--execute` command line option,
4. executes the files `file1`, `file2` . . . (given on the command line) in that order.

Note: `.singularrc` file(s) are an appropriate place for setting some default values of (command-line) options.

For example, a system administrator might remove the locally installed HTML version of the manual and put a `.singularrc` file with the following content

```
if (system("version") >= 1306) // assure backwards-compatibility
{
  system("--allow-net", 1);
}; // the last semicolon is important: otherwise no ">", but "." prompt
```

in the directory containing the Singular libraries, thereby allowing to fetch the HTML on-line help from the WWW home-site of Singular.

On the other hand, a single user might put a `.singularrc` with the following content

```
if (system("version") >= 1306) // assure backwards-compatibility
{
  if (! system("--emacs"))
  {
    // set default browser to info, unless we run within emacs
    system("--browser", "info");
  }
}; // the last semicolon is important: otherwise no ">", but "." prompt
```

in his home directory, which sets the default help browser to `info` (unless Singular is run within emacs) and thereby prevents the execution of the"global" `.singularrc` file installed by the system administrator (since the `.singularrc` file of the user is found before the "global" `.singularrc` file installed by the system administrator).

3.2 Emacs user interface

Besides running Singular in an ASCII-terminal, Singular might also be run within Emacs. Emacs (or, XEmacs which is very similar) is a powerful and freely available text editor, which, among others, provides a framework for the implementation of interactive user interfaces. Starting from version 1.3.6, Singular provides such an implementation, the so-called Singular Emacs mode, or Emacs user interface.

Generally, we recommend to use the Emacs interface, instead of the ASCII-terminal interface: The Emacs interface does not only provide everything the ASCII-terminal interface provides, but offers much more. Among others, it offers

- color highlighting
- truncation of long lines
- folding of input and output
- TAB-completion for help topics
- highlighting of matching parentheses
- key-bindings and interactive menus for most user interface commands and for basic Singular commands (such as loading of libraries and files)
- a mode for running interactive Singular demonstrations
- convenient ways to edit Singular input files
- interactive customization of nearly all aspects of the user-interface.

In order to use the Singular-Emacs interface you need to have Emacs version 20 or higher, or XEmacs version 20.3 or higher installed on your system. These editors can be downloaded for most hard- and software platforms, sources from either `http://www.gnu.org/software/emacs/emacs.html` (Emacs), or from `http://www.xemacs.org` (XEmacs). (Download of binaries depend on your OS). The differences between Emacs and XEmacs w.r.t. the Singular-Emacs interface are marginal – which editor to use is mainly a matter of personal preferences.

The simplest way to start-up Singular in its Emacs interface is by running the program `ESingular` which is contained in the Singular distribution. Alternatively, Singular can be started within an already running Emacs – see Section 3.2.2 [Running SINGULAR under Emacs], page 25 for details.

The next section gives a tutorial-like introduction to Emacs. This introductory section is followed by sections which explain the functionality of various aspects of the Emacs user interface in more detail: how to start/restart/kill Singular within Emacs, how to run an interactive demonstration, how to customize the Emacs user interface, etc. Finally, the 20 most important commands of the Emacs interface together with their key bindings are listed.

3.2.1 A quick guide to Emacs

This section gives a tutorial-like introduction to Emacs. Especially to users who are not familiar with Emacs, we recommend that they go through this section and try out the described features.

Emacs commands generally involve the `CONTROL` key (sometimes labeled `CTRL` or `CTL`) or the `META` key. On some keyboards, the `META` key is labeled `ALT` or `EDIT` or something else (for example, on Sun keyboards, the diamond key to the left of the space-bar is `META`). If there is no `META` key, the `ESC` key can be used, instead. Rather than writing out `META` or `CONTROL` each time we want to prefix a character, we will use the following abbreviations:

`C-<chr>` means hold the ⟨CONTROL⟩ key while typing the character `<chr>`. Thus, `C-f` would be: hold the ⟨CONTROL⟩ key and type `f`.

`M-<chr>` means hold the ⟨META⟩ key down while typing `<chr>`. If there is no ⟨META⟩ key, type ⟨ESC⟩, release it, then type the character `<chr>`.

For users new to Emacs, we highly recommend that they go through the interactive Emacs tutorial: type `C-h t` to start it.

For others, it is important to understand the following Emacs concepts:

window In Emacs terminology, a window refers to separate panes within the same window of the window system, and not to overlapping, separate windows. When using Singular within Emacs, extra windows may appear which display help or output from certain commands. The most important window commands are:

`C-x 1`	`File->Un-Split`	Un-Split window (i.e., kill other windows)
`C-x o`		Goto other window, i.e. move cursor into other window.

cursor and point

The location of the cursor in the text is also called "point". To paraphrase, the cursor shows on the screen where point is located in the text. Here is a summary of simple cursor-moving operations:

`C-f`	Move forward a character
`C-b`	Move backward a character
`M-f`	Move forward a word
`M-b`	Move backward a word
`C-a`	Move to the beginning of line
`C-e`	Move to the end of line

buffer

Any text you see in an Emacs window is always part of some buffer. For example, each file you are editing with Emacs is stored inside a buffer, but also Singular is running inside an Emacs buffer. Each buffer has a name: for example, the buffer of a file you edit usually has the same name as the file, Singular is running in a buffer which has the name `*singular*` (or, `*singular<2>*`, `*singular<3>*`, etc., if you have multiple Singular sessions within the same Emacs).

When you are asked for input to an Emacs command, the cursor moves to the bottom line of Emacs, i.e., to a special buffer, called the "minibuffer". Typing ⟨RETURN⟩ within the minibuffer, ends the input, typing ⟨SPACE⟩ within the minibuffer, lists all possible input values to the interactive Emacs command.

The most important buffer commands are

`C-x b`	Switch buffer
`C-x k`	Kill current buffer

Alternatively, you can switch to or kill buffers using the `Buffer` menu.

Executing commands

Emacs commands are executed by typing `M-x <command-name>` (remember that ⟨SPACE⟩ completes partial command names). Important and frequently used commands have short-cuts for their execution: Key bindings or even menu entries. For example, a file can be loaded with `M-x load-file`, or `C-x C-f`, or with the `File->Open` menu.

How to exit

To end the Emacs (and, Singular) session, type `C-x C-c` (two characters), or use the `File -> Exit` menu.

When Emacs hangs

If Emacs stops responding to your commands, you can stop it safely by typing `C-g`, or, if this fails, by typing `C-]`.

More help

Nearly all aspects of Emacs are very well documented: type `C-h` and then a character saying what kind of help you want. For example, typing `C-h i` enters the `Info` documentation browser.

Using the mouse

Emacs is fully integrated with the mouse. In particular, clicking the right mouse button brings up a pop-up menu which usually contains a few commonly used commands.

3.2.2 Running SINGULAR under Emacs

There are two ways to start the Singular Emacs interface: Typing `ESingular` instead of `Singular` on the command shell launches a new Emacs process, initializes the interface and runs Singular within Emacs. The other way is to start the interface in an already running Emacs, by typing `M-x singular` inside Emacs. This initializes the interface and runs Singular within Emacs. Both ways are described in more detail below.

Note: To properly run the Emacs interface, several files are needed which usually reside in the `emacs` subdirectory of your Singular distribution. This directory is called singular-emacs-home-directory in the following.

Starting the interface using ESingular

As mentioned above, `ESingular` is an "out-of-the-box" solution: You don't have to add special things to your `.emacs` startup file to initialize the interface; everything is done for you in a special file called `.emacs-singular` (which comes along with the Singular distribution and resides in the singular-emacs-home-directory) which is automatically loaded on Emacs startup (and the loading of the `.emacs` file is automatically suppressed).

The customizable variables of the Singular Emacs interface are set to defaults which give the novice user a very shell like feeling of the interface. Nevertheless, these default settings can be changed, see Section 3.2.4 [Customization of the Emacs interface], page 27. Besides other Emacs initializations, such as fontification or blinking parentheses, a new menu item called `Singular` is added to the main menu, providing menu items for starting Singular. On XEmacs, a button starting Singular is added to the main toolbar.

The Singular interface is started automatically; once you see a buffer called `*singular*` and the Singular prompt, you are ready to start your Singular session.

`ESingular` inherits all `Singular` options. For a description of all these options, see Section 3.1.6 [Command line options], page 19. Additionally there are the following options which are special to `ESingular`:

command-line option / environment variable	functionality

`--emacs=EMACS`
`ESINGULAR_EMACS`
	Use `EMACS` as Emacs program to run the Singular Emacs interface, where `EMACS` may e.g. be emacs or xemacs.

`--emacs-dir=DIR`
`ESINGULAR_EMACS_DIR`
	Set the singular-emacs-home-directory, which is the directory where singular.el can be found, to `DIR`.

`--emacs-load=FILE`
`ESINGULAR_EMACS_LOAD`
	Load `FILE` on Emacs start-up, instead of the default load file.

`--singular=PROG`
`ESINGULAR_SINGULAR`
	Start `PROG` as Singular program within Emacs

Notice that values of these options can also be given by setting the above mentioned environment variables (where values given as command-line arguments take priority over values given by environment variables).

Starting the interface within a running Emacs

If you are a more experienced Emacs user and you already have your own local `.emacs` startup file, you might want to start the interface out of your running Emacs without using `ESingular`. For this, you should add the following lisp code to your `.emacs` file:

```
     (setq load-path (cons "<singular-emacs-home-directory>" load-path))
     (autoload 'singular "singular"
       "Start Singular using default values." t)
     (autoload 'singular-other "singular"
       "Ask for arguments and start Singular." t)
```

Then typing `M-x singular` in a running Emacs session initializes the interface in a new buffer and launches a Singular process. The Singular prompt comes up and you are ready to start your Singular session.

It is a good idea to take a look at the (well documented) file `.emacs-singular` in the singular-emacs-home-directory, which comes along with the distribution. In it you find some useful initializations of the Singular interface as well as some lisp code, which, for example, adds a button to the XEmacs toolbar. Some of this code might be useful for your `.emacs` file, too. And if you are an Emacs wizard, it is of course a good idea to take a look at `singular.el` in the singular-emacs-home-directory.

CYGWIN and ESingular

X11 server install `xlaunch`, `emacs-X11`, `xterm` and all dependencies. Create with `xlaunch` a startup file for the X-server which also starts the client `xterm`. From that one can start ESingular.

fork problems The simplest way to overcome fork problem is to run `/usr/bin/rebase-trigger full`, then stop all Cygwin processes and services, and then run `setup-x86.exe`. The _autorebase postinstall script will then take care of the rebase. Occasionally it is necessary to reboot the computer before doing this.

Starting, interrupting and stopping SINGULAR

There are the following commands to start and stop Singular:

- `singular-other` (or menu `Singular`, item `Start...`)

  Starts a Singular process and asks for the following four parameters in the minibuffer area:

  1. The Singular executable. This can either be a file name with complete path, e.g., `/local/bin/Singular`. Then exactly this executable is started. The path may contain the character `~` denoting your home directory. Or it can be the name of a command without path, e.g., `Singular`. Then the executable is searched for in your `$PATH` environment variable.
  2. The default working directory. This is the path to an existing directory, e.g., `~/work`. The current directory is set to this directory before Singular is started.
  3. Command line options. You can set any Singular command line option (see Section 3.1.6 [Command line options], page 19).
  4. The buffer name. You can specify the name of the buffer the interface is running in.

- `singular` (or menu `Singular`, item `Start default`)

  Starts Singular with default settings for the executable, the working directory, command line switches, and the buffer name. You can customize this default settings, see Section 3.2.4 [Customization of the Emacs interface], page 27.

- `singular-exit-singular` (bound to `C-c $` or menu `Singular`, item `Exit`)

  Kills the running Singular process of the current buffer (but does not kill the buffer). Once you have killed a Singular process you can start a new one in the same buffer with the command `singular` (or select the item `Start default` of the `Singular` menu).

- `singular-restart` (bound to `C-c C-r` or menu `Singular`, item `Restart`)

  Kills the running SINGULAR process of the current buffer and starts a new process in the same buffer with exactly the same command line arguments as before.
- `singular-control-c` (bound to `C-c C-c` or menu `Singular`, item `Interrupt`)

  Interrupt the SINGULAR process running in the current buffer. Asks whether to (`a`)bort the current SINGULAR command, (`q`)uit or (`r`)estart the current SINGULAR process, or (`c`)ontinue without doing anything (default).

Whenever a SINGULAR process is started within the Emacs interface, the contents of a special startup file (by default `~/.emacs-singularrc`) is pasted as input to SINGULAR at the very end of the usual startup sequence (see Section 3.1.7 [Startup sequence], page 22). The name of the startup file can be changed, see Section 3.2.4 [Customization of the Emacs interface], page 27.

3.2.3 Demo mode

The Emacs interface can be used to run interactive SINGULAR demonstrations. A demonstration is started by loading a so-called SINGULAR demo file with the Emacs command `singular-demo-load`, bound to `C-c C-d`, or with the menu `Commands->Load Demo`.

A SINGULAR demo file should consist of SINGULAR commands separated by blank lines. When running a demo, the input up to the next blank line is echoed to the screen. Hitting ⟨RETURN⟩ executes the echoed commands and shows their output. Hitting ⟨RETURN⟩ again, echos the next commands to the screen, and so on, until all commands of the demo file are executed. While running a demo, you can execute other commands on the SINGULAR prompt: the next input from the demo file is then echoed again, if you hit ⟨RETURN⟩ on an empty input line.

A SINGULAR demo can prematurely be exited by either starting another demo, or by executing the Emacs command `singular-demo-exit` (menu: `Commands->Exit Demo`).

Some aspects of running SINGULAR demos can be customized. See Section 3.2.4 [Customization of the Emacs interface], page 27, for more info.

3.2.4 Customization of the Emacs interface

Emacs provides a convenient interface to customize the behavior of Emacs and the SINGULAR Emacs interface for your own needs. You enter the customize environment by either calling `M-x customize` (on XEmacs you afterwards have to enter `emacs` in the minibuffer area) or by selecting the menu item `Options->Customize->Emacs...` for XEmacs, and the menu item `Help->Customize->Toplevel Customization Group` for Emacs, resp. A brief introduction to the customization mode comes up with the customization buffer. All customizable parameters are hierarchically grouped and you can browse through all these groups and change the values of the parameters using the mouse. At the end you can safe your settings to a file making your changes permanent.

To change the settings of the SINGULAR Emacs interface you can either select the item `Preferences` of the `Singular` menu, call `M-x customize-group` and give the argument `singular-interactive` in the minibuffer area, or browse from the top-level customization group through the path `External->Singular->Singular interactive`.

The SINGULAR interface customization buffer is divided into four groups:

- Singular Faces

  Here you can specify various faces used if font-lock-mode is enabled (which, by default, is).

- Singular Sections And Foldings

  Here you can specify special faces for SINGULAR input and output and change the text used as replacement for folded sections.

  For doing this, you also might find handy the function `customize-face-at-point`, which lets you customize the face at the current position of point. This function is automatically defined if you run `ESingular`). Otherwise, you should add its definition (see below) to your personal `.emacs` file.
- Singular Interactive Miscellaneous

  Here you can specify various things such as the behavior of the cursor keys, the name of the special SINGULAR startup file, the appearance of the help window, or the default values for the `singular` command.
- Singular Demo Mode

  Here you can specify how chunks of the demo file are divided, or specify a default directory for demo files.

When you run `ESingular`, the settings of customized variables are saved in the file `$HOME/.emacs-singular-cust`. Otherwise, the settings are appended to your `.emacs` file. Among others, this means that the customized settings of `ESingular` are not automatically taken over by a "normal" Emacs, and vice versa.

3.2.5 Editing SINGULAR input files with Emacs

Since SINGULAR's programming language is similar to C, you should use the Emacs C/C++-mode to edit SINGULAR input files and SINGULAR libraries. Among others, this Emacs mode provides automatic indentation, line-breaking and keyword highlighting.

When running `ESingular`, the C/C++-mode is automatically turned on whenever a file with the suffix `.sing`, or `.lib` is loaded.

For Emacs sessions which were not started by `ESingular`, you should add the following to your `.emacs` file:

```
;; turn on c++-mode for files ending in ".sing" and ".lib"
(setq auto-mode-alist (cons '("\\.sing\\'" . c++-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.lib\\'" .  c++-mode) auto-mode-alist))
;; turn-on fontification for c++-mode
(add-hook 'c++-mode-hook
         (function (lambda () (font-lock-mode 1))))
;; turn on aut-new line and hungry-delete
(add-hook 'c++-mode-hook
          (function (lambda () (c-toggle-auto-hungry-state 1))))
;; a handy function for customization
(defun customize-face-at-point ()
  "Customize face which point is at."
  (interactive)
  (let ((face (get-text-property (point) 'face)))
    (if face
        (customize-face face)
      (message "No face defined at point"))))
```

Notice that you can change the default settings for source-code highlighting (colors, fonts, etc.) by customizing the respective faces using the `Customize` feature of Emacs. For doing this, you might find handy the above given function `customize-face-at-point`, which lets you customize the face of the current position of point (this function is automatically defined if you run `ESingular`).

3.2.6 Top 20 Emacs commands

Here is a list of the 20 probably most useful commands when using the Singular Emacs interface.

Starting and stopping of Singular:

- `singular` (menu `Singular->Start Default...`): starts Singular using default arguments.
- `singular-other` (menu `Singular->Start`): starts Singular asking for several arguments in the minibuffer area.
- `singular-exit` (key `C-c $` or menu `Singular->Exit`): kills the Singular process running in the current buffer (but does not kill the buffer).
- `singular-restart` (key `C-c C-r` or menu `Singular->Restart`): kills the Singular process running in the current buffer and starts a new Singular process with exactly the same arguments as before.

Editing input and output:

- `singular-beginning-of-line` (key `C-a`): moves point to beginning of line, then skips past the Singular prompt, if any.
- `singular-toggle-truncate-lines` (key `C-c C-t` or menu `Commands->Truncate lines`): toggles whether long lines should be truncated or not. If lines are not truncated, the commands `singular-scroll-left` and `singular-scroll-right` are useful to scroll left and right, resp.
- `singular-dynamic-complete` (key `TAB`): performs context specific completion. If point is inside a string, file name completion is done. If point is at the end of a help command (i.e., `help` or `?`), completion on Singular help topics is done. If point is at the end of an example command (i.e., `example`), completion is done on Singular examples. In all other cases, completion on Singular commands is done.
- `singular-folding-toggle-fold-latest-output` (key `C-c C-o` or menu `Commands->Fold/ Unfold Latest Output`): toggles folding of the latest output section. If your last Singular command produced a huge output, simply type `C-c C-o` and it will be replaced by a single line.
- `singular-folding-toggle-fold-at-point` (key `C-c C-f` or menu `Commands->Fold/Unfold At Point`): toggles folding of the section the point currently is in.
- `singular-folding-fold-all-output` (menu `Commands->Fold All Output`): folds all Singular output, replacing each output section by a single line.
- `singular-folding-unfold-all-output` (menu `Commands->Unfold All Output`): unfolds all Singular output sections showing their true contents.

Loading of files and Singular demo mode:

- `singular-load-library` (key `C-c C-l` or menu `Commands->Libraries->other...`): asks for a standard library name or a library file in the minibuffer (hit `TAB` for completion) and loads the library into Singular. The submenu `Libraries` of the `Commands` menu also provides a separate menu item for each standard library.
- `singular-load-file` (key `C-c <` or menu `Commands->Load File...`): asks for a file name in the minibuffer (which is expanded using `expand-file-name` if given a prefix argument) and loads the file into Singular.
- `singular-demo-load` (key `C-c C-d` or menu `Commands->Load Demo...`): asks for a file name of a Singular demo file in the minibuffer area (hit `SPACE` for completion) and enters the Singular demo mode showing the first chunk of the demo.
- `singular-demo-exit` (menu `Commands->Exit Demo`): exits from Singular demo mode and cleans up everything that is left from the demo.

Help and Customization:

- `singular-help` (key `C-h C-s` or menu `Singular->Singular Help`): asks for a SINGULAR help topic in the minibuffer (hit `TAB` for completion) and shows the help text in a separate buffer.
- `singular-example` (key `C-c C-e` or menu `Singular->Singular Example`): asks for a SINGULAR command in the minibuffer (hit `TAB` for completion) and executes the example of this command in the current SINGULAR buffer.
- `customize-group` (menu `Singular->Preferences`): enters the customization group of the SINGULAR Emacs interface. (If called via `M-x customize-group` give argument `singular-interactive` in the minibuffer area.)

3.3 Rings and orderings

All non-trivial algorithms in SINGULAR require the prior definition of a ring. Such a ring can be

1. a polynomial ring over a field,
2. a polynomial ring over a ring
3. a localization of 1.
4. a quotient ring by an ideal of 1. or 2.,
5. a tensor product of 1. or 2.

Except for quotient rings, all of these rings are realized by choosing a coefficient field, ring variables, and an appropriate global or local monomial ordering on the ring variables. See Section 3.3.3 [Term orderings], page 34, Appendix C [Mathematical background], page 774.

The coefficient field of the rings may be

1. the field of rational numbers Q (`QQ`),
2. finite fields Z/p, p a prime ≤ 2147483647,
3. finite fields GF(p^n) with p^n elements, p a prime, $p^n \leq 2^{16}$,
4. transcendental extension of Q or Z/p ,
5. simple algebraic extension of Q or Z/p ,
6. the field of real numbers represented by floating point numbers of a user defined precision,
7. the field of complex numbers represented by (pairs of) floating point numbers of a user defined precision,
8. the ring of integers (`ZZ`),
9. finite rings Z/m with $m \in Z$.

In case of coefficient rings, which are not fields (i.e. Z and Z/ma), only the following functions are guaranteed to work:

- basic polynomial arithmetic, i.e. addition, multiplication, exponentiation
- std, i.e. computing standard bases (and related: syz, etc.)
- interred
- reduce

Throughout this manual, the current active ring in SINGULAR is called basering. The reserved name `basering` in SINGULAR is an alias for the current active ring. The basering can be set by declaring a new ring as described in the following subsections or by using the commands `setring` and `keepring`. See Section 5.2.11 [keepring], page 298, Section 5.1.141 [setring], page 260.

Objects of ring dependent types are local to a ring. To access them after a change of the basering they have to be mapped using `map` or by the functions `imap` or `fetch`. See Section 3.5.4 [Objects], page 46, Section 5.1.38 [fetch], page 182, Section 5.1.59 [imap], page 198, Section 4.12 [map], page 106.

All changes of the basering in a procedure are local to this procedure unless a `keepring` command is used as the last statement of the procedure. See Section 3.7 [Procedures], page 50, Section 5.2.11 [keepring], page 298.

3.3.1 Examples of ring declarations

The exact syntax of a ring declaration is given in the next two subsections; this subsection lists some examples first. Note that the chosen ordering implies that a unit-elements of the ring will be among the elements with leading monomial 1. For more information, see Section B.2 [Monomial orderings], page 768.

Every floating point number in a ring consists of two parts, which may be chosen by the user. The leading part represents the number and the rest is for numerical stability. Two numbers with a difference only in the rest will be regarded equal.

- the ring $Z/32003[x, y, z]$ with degree reverse lexicographical ordering. The exact ring declaration may be omitted in the first example since this is the default ring:

  ```
  ring r1;
  ring r2 = 32003,(x,y,z),dp;
  ring r3=(ZZ/32003)[x,y,z];
  ring r4 = (ZZ/32003),(x,y,z),dp;
  ```

- similar examples with indexed variables. The ring variables of r1 are going to be x(1)..x(10); in r2 they will be x(1)(1), x(1)(2), ..., x(1)(8), x(2)(1), ..., x(5)(8):

  ```
  ring r1 = 32003,(x(1..10)),dp;
  ring r2 = 32003,(x(1..5)(1..8)),dp;
  ring r3 = (ZZ/32003)[x(1..5)(1..8)];
  ring r4 = (ZZ/32003),(x(1..5)(1..8)),dp;
  ```

- the ring $Q[a, b, c, d]$ with lexicographical ordering:

  ```
  ring r1 = 0,(a,b,c,d),lp;
  ring r2 = QQ,(a,b,c,d),lp;
  ```

- the ring $Z/7[x, y, z]$ with local degree reverse lexicographical ordering. The non-prime 10 is converted to the next lower prime in the second example:

  ```
  ring r1 = 7,(x,y,z),ds;
  ring r2 = 10,(x,y,z),ds;
  ring r3 = (ZZ/7),(x,y,z),ds;
  ```

- the ring $Z/7[x_1, \ldots, x_6]$ with lexicographical ordering for x_1, x_2, x_3 and degree reverse lexicographical ordering for x_4, x_5, x_6:

  ```
  ring r1 = 7,(x(1..6)),(lp(3),dp);
  ring r2 = (ZZ/7),(x(1..6)),(lp(3),dp);
  ```

- the localization of $(Q[a, b, c])[x, y, z]$ at the maximal ideal
  (x, y, z) :

  ```
  ring r1 = 0,(x,y,z,a,b,c),(ds(3), dp(3));
  ring r2 = QQ,(x,y,z,a,b,c),(ds(3), dp(3));
  ```

- the ring $Q[x, y, z]$ with weighted reverse lexicographical ordering. The variables x , y , and z have the weights 2, 1, and 3, respectively, and vectors are first ordered by components (in descending order) and then by monomials:

  ```
  ring r1 = 0,(x,y,z),(c,wp(2,1,3));
  ring r2 = QQ,(x,y,z),(c,wp(2,1,3));
  ```

  For ascending component order, the component ordering `C` has to be used.

- the ring $K[x,y,z]$, where $K = Z/7(a,b,c)$ denotes the transcendental extension of $Z/7$ by a , b and c with degree lexicographical ordering:
  ```
  ring r = (7,a,b,c),(x,y,z),Dp;
  ```
- the ring $K[x,y,z]$, where $K = Z/7[a]$ denotes the algebraic extension of degree 2 of $Z/7$ by a. In other words, K is the finite field with 49 elements. In the first case, a denotes an algebraic element over $Z/7$ with minimal polynomial $\mu_a = a^2 + a + 3$, in the second case, a

  refers to some generator of the cyclic group of units of K :
  ```
  ring r = (7,a),(x,y,z),dp; minpoly = a^2+a+3;
  ring r = (7^2,a),(x,y,z),dp;
  ```
- the ring $R[x,y,z]$, where R denotes the field of real numbers represented by simple precision floating point numbers. This is a special case:
  ```
  ring r = real,(x,y,z),dp;
  ```
- the ring $R[x,y,z]$, where R denotes the field of real numbers represented by floating point numbers of 50 valid decimal digits and the same number of digits for the rest:
  ```
  ring r = (real,50),(x,y,z),dp;
  ```
- the ring $R[x,y,z]$, where R denotes the field of real numbers represented by floating point numbers of 10 valid decimal digits and with 50 digits for the rest:
  ```
  ring r = (real,10,50),(x,y,z),dp;
  ```
- the ring $R(j)[x,y,z]$, where R denotes the field of real numbers represented by floating point numbers of 30 valid decimal digits and the same number for the rest. j denotes the imaginary unit.
  ```
  ring r = (complex,30,j),(x,y,z),dp;
  ```
- the ring $R(i)[x,y,z]$, where R denotes the field of real numbers represented by floating point numbers of 6 valid decimal digits and the same number for the rest. i is the default for the imaginary unit.
  ```
  ring r = complex,(x,y,z),dp;
  ```
- the quotient ring $Z/7[x,y,z]$ modulo the square of the maximal ideal (x,y,z) :
  ```
  ring R = 7,(x,y,z), dp;
  qring r = std(maxideal(2));
  ```
- the ring $Z[x,y,z]$:
  ```
  ring R = integer,(x,y,z), dp;
  ```
- the ring $Z/6^3[x,y,z]$:
  ```
  ring R = (integer, 6, 3),(x,y,z), dp;
  ```
- the ring $Z/100[x,y,z]$:
  ```
  ring R = (integer, 100),(x,y,z), dp;
  ```

3.3.2 General syntax of a ring declaration

Rings

Syntax: `ring` name `=` (coefficients)`,` `(` names_of_ring_variables `),` `(` ordering `);` or
`ring` name `=` cring `[` names_of_ring_variables `]`

Default: `(ZZ/32003)[x,y,z];`

Purpose: declares a ring and sets it as the current basering. The second form sets the ordering to `(dp,C)`. `cring` stands currently for `QQ` (the rationals), `ZZ` (the integers) or `(ZZ/m)` (the field (m prime and <2147483648) resp. ring of the integers modulo m).

The coefficients (for the first form) are given by one of the following:

1. a `cring` as given above
2. a non-negative int_expression less than 2147483648 (2^31).
   The int_expression should either be 0, specifying the field of rational numbers Q, or a prime number p, specifying the finite field with p elements. If it is not a prime number, int_expression is converted to the next lower prime number.
3. an expression_list of an int_expression and one or more names.
   The int_expression specifies the characteristic of the coefficient field as described above. The names are used as parameters in transcendental or algebraic extensions of the coefficient field. Algebraic extensions are implemented for one parameter only. In this case, a minimal polynomial has to be defined by an assignment to `minpoly`. See Section 5.3.3 [minpoly], page 303.
4. an expression_list of an int_expression and a name.
   The int_expression has to be a prime number p to the power of a positive integer n. This defines the Galois field GF(p^n) with p^n elements, where p^n has to be less than or equal to 2^{15}. The given name refers to a primitive element of GF(p^n) generating the multiplicative group. Due to a different internal representation, the arithmetic operations in these coefficient fields are faster than arithmetic operations in algebraic extensions as described above.
5. an expression_list of the name `real` and two optional int_expressions determining the precision in decimal digits and the size for the stabilizing rest. The default for the rest is the same size as for the representation. An exception is the name `real` without any integers. These numbers are implemented as machine floating point numbers of single precision. Note that computations over all these fields are not exact.
6. an expression_list of the name `complex`, two optional int_expression and a name. This specifies the field of complex numbers represented by floating point numbers with a precision similar to `real`. An expression_list without int_expression defines a precision and rest with length 6. The name of the imaginary unit is given by the last parameter. Note that computations over these fields are not exact.
7. an expression_list with the name `integer`. This specifies the ring of integers.
8. an expression_list with the name `integer` and one subsequent int_expression. This specifies the ring of integers modulo the given int_expression.
9. an expression_list with the name `integer` and two int_expressions `b` and `e`. This specifies the ring of integers modulo b^e. If `b = 2` and `e < int_bit_size` an optimized implementation is used.

'names_of_ring_variables' is a list of names or indexed names.

'ordering' is a list of block orderings where each block ordering is either

1. `lp`, `dp`, `Dp`, `ls`, `ds`, or `Ds` optionally followed by a size parameter in parentheses.
2. `wp`, `Wp`, `ws`, `Ws`, or `a` followed by a weight vector given as an intvec_expression in parentheses.
3. `M` followed by an intmat_expression in parentheses.
4. `c` or `C`.

For the definition of the orderings, see Section B.2 [Monomial orderings], page 768.

If one of coefficients, names_of_ring_variables, and ordering consists of only one entry, the parentheses around this entry may be omitted.

Quotient rings

Syntax: `qring` name = ideal_expression ;

Default: none

Purpose: declares a quotient ring as the basering modulo ideal_expression, and sets it as current basering.

ideal_expression has to be represented by a standard basis.

The most convenient way to map objects from a ring to its quotient ring and vice versa is to use the `fetch` function (see Section 5.1.38 [fetch], page 182).

SINGULAR computes in a quotient ring as long as possible with the given representative of a polynomial, say, `f`. I.e., it usually does not reduce `f` w.r.t. the quotient ideal. This is only done when necessary during standard bases computations or by an explicit reduction using the command `reduce(f, std(0))` (see Section 5.1.131 [reduce], page 251).

Operations based on standard bases (e.g. `std`,`groebner`, etc., `reduce`) and functions which require a standard basis (e.g. `dim`,`hilb`, etc.) operated with the residue classes; all others on the polynomial objects.

Example:

```
  // definition and usage:
  ring r=(ZZ/32003)[x,y];
  poly f=x3+yx2+3y+4;
  qring q=std(maxideal(2));
  basering;
↦ // coefficients: ZZ/32003
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=y2
↦ _[2]=xy
↦ _[3]=x2
  poly g=fetch(r, f);
  g;
↦ x3+x2y+3y+4
  reduce(g,std(0));
↦ 3y+4
  // polynomial and residue class:
  ring R=QQ[x,y];
  qring Q=std(y);
  poly p1=x;
  poly p2=x+y;
  // comparing polynomial objects:
  p1==p2;
↦ 0
  // comparing residue classes:
  reduce(p1,std(0))==reduce(p2,std(0));
↦ 1
```

3.3.3 Term orderings

Any polynomial (resp. vector) in SINGULAR is ordered w.r.t. a term ordering (or, monomial ordering), which has to be specified together with the declaration of a ring. SINGULAR stores and displays a polynomial (resp. vector) w.r.t. this ordering, i.e., the greatest monomial (also called the leading monomial) is the first one appearing in the output polynomial, and the smallest monomial is the last one.

Remark: The novice user should generally use the ordering `dp` for computations in the polynomial ring $K[x_1, \ldots, x_n]$, resp. `ds` for computations in the localization $\mathrm{Loc}_{(x)}K[x_1, \ldots, x_n]$). For more details, see Appendix B [Polynomial data], page 767.

In a ring declaration, SINGULAR offers the following orderings (but see also Section B.2 [Monomial orderings], page 768):

1. Global orderings

   `lp` lexicographical ordering

   `rp` inverse lexicographical ordering, i.e. a lexicographical ordering from the right with 1 < x_1 < ... <x_n
   (should not be used as it reverses the "natural" x_1 > ... > x_n, reorder the variables instead)

   `dp` degree reverse lexicographical ordering

   `Dp` degree lexicographical ordering

   `wp(` intvec_expression `)`
   weighted reverse lexicographical ordering; the weight vector is expected to consist of positive integers only.

   `Wp(` intvec_expression `)`
   weighted lexicographical ordering; the weight vector is expected to consist of positive integers only.

   `Ip` degree inverse lexicographical ordering, i.e.sorting by degree, followed by a lexicographical ordering from the right with 1 < x_1 < ... <x_n
   (should not be used as it reverses the "natural" x_1 > ... > x_n, reorder the variables instead)

   Global orderings are well-orderings, i.e., $1 < x$ for each ring variable x . They are denoted by a `p` as the second character in their name.

2. Local orderings

   `ls` negative lexicographical ordering

   `rs` negative inverse lexicographical ordering, i.e. a lexicographical ordering from the right
   (should not be used as it reverses the "natural" x_1 < ... < x_n, reorder the variables instead)

   `ds` negative degree reverse lexicographical ordering

   `Ds` negative degree lexicographical ordering

   `ws(` intvec_expression `)`
   (general) weighted reverse lexicographical ordering; the first element of the weight vector has to be non-zero.

   `Ws(` intvec_expression `)`
   (general) weighted lexicographical ordering; the first element of the weight vector has to be non-zero.

   Local orderings are not well-orderings. They are denoted by an `s` as the second character in their name.

3. Matrix orderings

   `M(` intmat_expression `)`

   intmat_expression has to be an invertible square matrix

   Using matrix orderings, SINGULAR can compute standard bases w.r.t. any monomial ordering which is compatible with the natural semi-group structure on the monomials. In practice, the predefined global and local orderings together with the block orderings should be sufficient in most cases. These orderings are faster than their corresponding matrix orderings since evaluation of a matrix ordering is more time consuming.

4. Extra weight vector

   `a(` intvec_expression `)`

   `aa(` intvec_expression `)`

   `am(` intvec_expression `)`

   an extra weight vector `a(` intvec_expression `)`, may precede any monomial ordering.
   `aa` allows larger degrees.
   `am` allows degrees for module generators (see example)

   Example for ordering `am`

   ```
   ring r=QQ,(x,y,z),(am(1,1,1,4,1,6),dp);
   r;
   ↦ // coefficients: QQ
   ↦ // number of vars : 3
   ↦ //        block   1 : ordering am
   ↦ //                  : names    x y z
   ↦ //                  : weights  1 1 1
   ↦ //                  : 3 module weights  4 1 6
   ↦ //        block   2 : ordering dp
   ↦ //                  : names    x y z
   ↦ //        block   3 : ordering C
   [x,y,z];
   ↦ z*gen(3)+x*gen(1)+y*gen(2)
   deg([x]);
   ↦ 5
   deg([0,y]);
   ↦ 2
   deg([0,0,z]);
   ↦ 7
   deg([0,0,0,x]);
   ↦ 1
   ```

5. Product ordering

   `(` ordering [`(` int_expression `)`]`, ... )`

   any of the above orderings and the extra weight vector may be combined to yield product or block orderings

   The orderings `lp`, `dp`, `Dp`, `ls`, `ds`, `Ds` and `rp` may be followed by an int_expression in parentheses giving the size of the block. For the last block the size is calculated automatically. For weighted orderings, the size of the block is given by the size of the weight vector. The same holds analogously for matrix orderings.

6. Module orderings

`( ordering, ..., C )`
`( ordering, ..., c )`
sort polynomial vectors by the monomial ordering first, then by components

`( C, ordering, ... )`
`( c, ordering, ... )`
sort polynomial vectors by components first, then by the monomial ordering

Here a capital `C` sorts generators in ascending order, i.e., `gen(1) < gen(2) < ...`. A small `c` sorts in descending order, i.e., `gen(1) > gen(2) > ...`. It is not necessary to specify the module ordering explicitly since `( ordering, ..., C )` is the default.

In fact, `c` or `C` may be specified anywhere in a product ordering specification, not only at its beginning or end. All monomial block orderings preceding the component ordering have higher precedence, all monomial block orderings following after it have lower precedence.

For a mathematical description of these orderings, see Appendix B [Polynomial data], page 767.

3.3.4 Coefficient rings

Singular supports coefficient ranges which are not fields, i.e. the integers Z and the finite rings Z/n for a number `n`. These coefficient rings were implemented in Singular 3.0.5 and at the moment only limited functionality is available.

p-adic numbers

The p-adic integers Z_p are the projective limit of the finite rings Z/p^n for `n` to infinity. Therefore, computations in this ring can be approximated by computations in Z/p^n for large `n`.

3.4 Implemented algorithms

The basic algorithm in Singular is a general standard basis algorithm for any monomial ordering which is compatible with the natural semi-group structure of the exponents. This includes well-orderings (Buchberger algorithm to compute a Groebner basis) and tangent cone orderings (Mora algorithm) as special cases.

Nonetheless, there are a lot of other important algorithms:

- Algorithms to compute the standard operations on ideals and modules: intersection, ideal quotient, elimination, etc.
- Different Syzygy algorithms and algorithms to compute free resolutions of modules.
- Combinatorial algorithms to compute dimensions, Hilbert series, multiplicities, etc.
- Algorithms for univariate and multivariate polynomial factorization, resultant and gcd computations.

Commands to compute standard bases

`facstd` Section 5.1.34 [facstd], page 179
computes a list of Groebner bases via the Factorizing Groebner Basis Algorithm, i.e., their intersection has the same radical as the original ideal. It need not be a Groebner basis of the given ideal.

The intersection of the zero-sets is the zero-set of the given ideal.

`fglm` Section 5.1.39 [fglm], page 183
computes a Groebner basis provided that a reduced Groebner basis w.r.t. another ordering is given.

Implements the so-called FGLM (Faugere, Gianni, Lazard, Mora) algorithm. The given ideal must be zero-dimensional.

`groebner` Section 5.1.53 [groebner], page 191
computes a standard resp. Groebner basis using a heuristically chosen method.

This is the preferred method to compute a standard resp. Groebner bases.

`mstd` Section 5.1.100 [mstd], page 227
computes a standard basis and a minimal set of generators.

`std` Section 5.1.151 [std], page 271
computes a standard resp. Groebner basis.

`stdfglm` Section 5.1.152 [stdfglm], page 272
computes a Groebner basis in a ring with a “difficult” ordering (e.g., lexicographical) via `std` w.r.t. a “simple” ordering and `fglm`.

The given ideal must be zero-dimensional.

`stdhilb` Section 5.1.153 [stdhilb], page 273
computes a Groebner basis in a ring with a “difficult” ordering (e.g., lexicographical) via `std` w.r.t. a “simple” ordering and a `std` computation guided by the Hilbert series.

Further processing of standard bases

The next commands require the input to be a standard basis.

`degree` Section 5.1.20 [degree], page 171
computes the (Krull) dimension, codimension and the multiplicity.

The result is only displayed on the screen.

`dim` Section 5.1.25 [dim], page 174
computes the dimension of the ideal resp. module.

`highcorner`
Section 5.1.55 [highcorner], page 194
computes the smallest monomial not contained in the ideal resp. module. The ideal resp. module has to be finite dimensional as a vector space over the ground field.

`hilb` Section 5.1.56 [hilb], page 195
computes the first, and resp. or, second Hilbert series of an ideal resp. module.

`kbase` Section 5.1.69 [kbase], page 205
computes a vector space basis (consisting of monomials) of the quotient of a ring by an ideal resp. of a free module by a submodule.

The ideal resp. module has to be finite dimensional as a vector space over the ground field and has to be represented by a standard basis w.r.t. the ring ordering.

`mult` Section 5.1.101 [mult], page 228
computes the degree of the monomial ideal resp. module generated by the leading monomials of the input.

`reduce` Section 5.1.131 [reduce], page 251
reduces a polynomial, vector, ideal or module to its normal form with respect to an ideal or module represented by a standard basis.

`vdim`
Section 5.1.168 [vdim], page 286
computes the vector space dimension of a ring (resp. free module) modulo an ideal (resp. module).

Commands to compute resolutions

`res`
Section 5.1.134 [res], page 253
computes a free resolution of an ideal or module using a heuristically chosen method. This is the preferred method to compute free resolutions of ideals or modules.

`fres`
Section 5.1.48 [fres], page 188
improved version of Section 5.1.149 [sres], page 269, computes a free resolution of an ideal or module using Schreyer's method. The input has to be a standard basis.

`lres`
Section 5.1.83 [lres], page 215
computes a free resolution of an ideal or module with LaScala's method. The input needs to be homogeneous.

`mres`
Section 5.1.98 [mres], page 225
computes a minimal free resolution of an ideal or module with the Syzygy method.

`sres`
Section 5.1.149 [sres], page 269
computes a free resolution of an ideal or module with Schreyer's method. The input has to be a standard basis.

`nres`
Section 5.1.106 [nres], page 232
computes a free resolution of an ideal or module with the standard basis method.

`syz`
Section 5.1.156 [syz], page 280
computes the first Syzygy (i.e., the module of relations of the given generators).

Further processing of resolutions

`betti`
Section 5.1.4 [betti], page 159
computes the graded Betti numbers of a module from a free resolution.

`minres`
Section 5.1.93 [minres], page 223
minimizes a free resolution of an ideal or module.

`regularity`
Section 5.1.132 [regularity], page 252
computes the regularity of a homogeneous ideal resp. module from a given minimal free resolution.

Processing of polynomials

`char_series`
Section 5.1.6 [char_series], page 161
computes characteristic sets of polynomial ideals.

`extgcd`
Section 5.1.33 [extgcd], page 178
computes the extended gcd of two polynomials.

This is implemented as extended Euclidean Algorithm, and applicable for univariate polynomials only.

`factorize`

Section 5.1.36 [factorize], page 180
computes factorization of univariate and multivariate polynomials into irreducible factors.

The most basic algorithm is univariate factorization in prime characteristic. The Cantor-Zassenhaus Algorithm is used in this case. For characteristic 0, a univariate Hensel-lifting is done to lift from prime characteristic to characteristic 0. For multivariate factorization in any characteristic, the problem is reduced to the univariate case first, then a multivariate Hensel-lifting is used to lift the univariate factorization.

Factorization of polynomials over algebraic extensions is provided by factoring the norm for univariate polynomials f (the gcd of f and the factors of the norm is a factorization of f) resp. by the extended Zassenhaus algorithm for multivariate polynomials.

`gcd`

Section 5.1.50 [gcd], page 190
computes greatest common divisors of univariate and multivariate polynomials.

In the univariate case NTL is used. For prime characteristic, a subresultant gcd is used. In characteristic 0, the EZGCD is used, except for a special case where a modular algorithm is used.

`resultant`

Section 5.1.136 [resultant], page 255
computes the resultant of two univariate polynomials using the subresultant algorithm.

Multivariate polynomials are considered as univariate polynomials in the main variable (which has to be specified by the user).

`vandermonde`

Section 5.1.164 [vandermonde], page 284
interpolates a polynomial from its values at several points

Matrix computations

`bareiss`

Section 5.1.3 [bareiss], page 158
implements sparse Gauss-Bareiss method for elimination (matrix triangularization) in arbitrary integral domains.

`det`

Section 5.1.23 [det], page 172
computes the determinant of a square matrix.

For matrices with integer entries a modular algorithm is used. For other domains the Gauss-Bareiss method is used.

`minor`

Section 5.1.92 [minor], page 221
computes all minors (=subdeterminants) of a given size for a matrix.

Numeric computations

`laguerre`

Section 5.1.74 [laguerre], page 208
computes all (complex) roots of a univariate polynomial

`uressolve`

Section 5.1.163 [uressolve], page 283
finds all roots of a 0-dimensional ideal with multivariate resultants

Controlling computations

`option` Section 5.1.111 [option], page 234
allows setting of options for manipulating the behaviour of computations (such as reduction strategies) and for showing protocol information indicating the progress of a computation.

3.5 The SINGULAR language

Singular interprets commands given interactively on the command line as well as given in the context of user-defined procedures. In fact, Singular makes no distinction between these two cases. Thus, Singular offers a powerful programming language as well as an easy-to-use command line interface without differences in syntax or semantics.

In the following, the basic language concepts such as commands, expressions, names, objects, etc., are discussed. See Section 3.7 [Procedures], page 50, and Section 3.8 [Libraries], page 55, for the concepts of procedures and libraries.

In many aspects, the Singular language is similar to the C programming language. For a description of some of the subtle differences, see Section 6.3 [Major differences to the C programming language], page 310.

Elements of the language

The major building blocks of the Singular language are expressions, commands, and control structures. The notion of expressions in the Singular and the C programming language are identical, whereas the notion of commands and control structures only roughly corresponds to C statements.

- An “expression” is a sequence of operators, functions, and operands that specifies a computation. An expression always results in a value of a specific type. See Chapter 4 [Data types], page 73, and its subsections (e.g., Section 4.17.2 [poly expressions], page 121), for information on how to build expressions.
- A “command” is either a declaration, an assignment, a call to a function without return value, or a print command. For detailed information, see Section 3.5.1 [General command syntax], page 41.
- “Control structures” determine the execution sequence of commands. Singular provides control structures for conditional execution (`if ... else`) and iteration (`for` and `while`). Commands may be grouped in pairs of `{ }` (curly brackets) to form blocks. See Section 5.2 [Control structures], page 290, for more information.

Other notational conventions

For user-defined functions, the notions of “procedure” and “function” are synonymous.

As already mentioned above, functions without return values are called commands. Furthermore, whenever convenient, the term “command” is used for a function, even if it does return a value.

3.5.1 General command syntax

In Singular a command is either a declaration, an assignment, a call to a function without return value, or a print command. The general form of a command is described in the following subsections.

Declaration

1. type name = expression ;
   declares a variable with the given name of the given type and assigns the expression as initial value to it. Expression is an expression of the specified type or one that can be converted to that type. See Section 3.5.5 [Type conversion and casting], page 46.
2. `alias` type name
   Introduces name as an alternative, read-only name for another variable_name. Can only be used in procedure headings to avoid copying large data.
3. type name_list = expression_list ;
   declares variables with the given names and assigns successively each expression of expression_list to the corresponding name of name_list. Both lists must be of the same length. Each expression in expression_list is an expression of the specified type or one that can be converted to that type. See Section 3.5.5 [Type conversion and casting], page 46.
4. type name ;
   declares a variable with the given name of the given type and assigns the default value of the specific type to it.

See Section 3.5.3 [Names], page 44, for more information on declarations. See Chapter 4 [Data types], page 73, for a description of all data types known to SINGULAR.

```
ring r;                    // the default ring
poly f,g = x^2+y^3,xy+z2; // the polynomials f=x^2+y^3 and g=x*y+z^2
ideal I = f,g;             // the ideal generated by f and g
matrix m[3][3];            // a 3 x 3 zero matrix
int i=2;                   // the integer i=2
```

Assignment

4. name = expression ;
   assigns expression to name.
5. name_list = expression_list ;
   assigns successively each expression of expression_list to the corresponding name of name_list. Both lists must be of the same length. This is not a simultaneous assignment. Thus, `f, g = g, f;` does not swap the values of `f` and `g`, but rather assigns `g` to both `f` and `g`.

A type conversion of the type of expression to the type of name must be possible. See Section 3.5.5 [Type conversion and casting], page 46.

An assignment itself does not yield a value. Hence, compound assignments like `i = j = k;` are not allowed and result in an error.

```
f = x^2 + y^2 ;       // overrides the old value of f
I = jacob(f);
f,g = I[1],x^2+y^2 ; // overrides the old values of f and g
```

Function without return value

6. function_name [(argument_list)] ;
   calls function function_name with arguments argument_list.

The function may have output (not to be confused with a return value of type string). See Section 5.1 [Functions], page 156. Functions without a return value are specified there to have a return type 'none'.

Some of these functions have to be called without parentheses, e.g., `help`, `LIB`.

```
  ring r;
  ideal i=x2+y2,x;
  i=std(i);
  degree(i);        // degree has no return value but prints output
↦ // dimension (proj.)  = 0
↦ // degree (proj.)   = 2
```

Print command

7. expression ;
   prints the value of an expression, for example, of a variable.

Use the function `print` (or the procedure `show` from inout.lib) to get a pretty output of various data types, e.g., matrix or intmat. See Section 5.1.120 [print], page 242.

```
  int i=2;
  i;
↦ 2
  intmat m[2][2]=1,7,10,0;
  print(m);
↦      1     7
↦     10     0
```

3.5.2 Special characters

The following characters and operators have special meanings:

`=` assignment

`{`, `}` parentheses for block programming

`(`, `)` in expressions, for indexed names and for argument lists

`[`, `]` access operator for strings, integer vectors, ideals, matrices, polynomials, resolutions, and lists. Used to build vectors of polynomials. Example: `s[3]`, `m[1,3]`, `i[1..3]`, `[f,g+x,0,0,1]`.

`+` addition operator

`++` increment operator

`-` subtraction operator

`--` decrement operator

`*` multiplication operator

`/` division operator. See Section 6.4 [Miscellaneous oddities], page 313, for the difference between the division operators `/` and `div`.

`%` modulo operator (`mod` is an alias to `%`): result is always non-negative

`^` or `**` exponentiation operator

`==` comparison operator equal

`!=` or `<>` comparison operator not equal

`>=` comparison operator larger than or equal to

`>` comparison operator larger

`<=` comparison operator smaller than or equal to

`<` comparison operator smaller. Also used for file input. See Section 5.1.41 [filecmd], page 184.

`!` boolean operator not

`&&` boolean operator and

`||` boolean operator or

`"` delimiter for string constants

‘ delimiter for name substitution

`?` synonym for `help`

`//` comment delimiter. Comment extends to the end of the line.

`/*` comment delimiter. Starts a comment which ends with `*/`.

`*/` comment delimiter. Ends a comment which starts with `/*`.

`;` statement separator

`,` separator for expression lists and function arguments

`\` escape character for `"` and `\` within strings

`..` interval specifier returning intvec. E.g., `1..3` which is equivalent to the intvec `1, 2, 3`.

`:` repeated entry. E.g., `3:5` generates an intvec of length 5 with constant entries 3, i.e., (3, 3, 3, 3, 3).

`::` accessor for package members. E.g., `MyPackage::i` accesses variable `i` in package `MyPackage`.

`_` value of expression displayed last

`~` breakpoint in procedures

`#` list of parameters in procedures without explicit parameter list

`$` terminates Singular

3.5.3 Names

Singular is a strongly typed language. This means that all names (= identifiers) have to be declared prior to their use. For the general syntax of a declaration, see the description of declaration commands (see Section 3.5.1 [General command syntax], page 41).

See Chapter 4 [Data types], page 73, for a description of Singular's data types. See Section 5.1.161 [typeof], page 282, for a short overview of possible types. To get information on a name and the object named by it, the `type` command may be used (see Section 5.1.160 [type], page 282).

It is possible to redefine an already existing name if doing so does not change its type. A redefinition first sets the variable to the default value and then computes the expression. The difference between redefining and overriding a variable is shown in the following example:

```
  int i=3;
  i=i+1;        // overriding
  i;
↦ 4
  int i=i+1;    // redefinition
↦ // ** redefining i (  int i=i+1;    // redefinition) ./examples/Names.sin\
   g:4
```

```
    i;
  ↦ 1
```

User defined names should start with a letter and consist of letters and digits only. As an exception to this rule, the characters `@`, and `_` may be used as part of a name, too (`@` as the first letter is reserved for purposes of library routines). Capital and small letters are distinguished. Indexed names are built as a name followed by an int_expression in parentheses. A list of indexed names can be built as a name followed by an intvec_expression in parentheses. For multi-indices, append an int_expression in parentheses to an indexed name. An alternative multi-index construction is `name_prefix(` index_1, index_2,...) where the `name_prefix` must be an undefined name.

```
    ring R;
    int n=3;
    ideal j(3);
    ideal j(n);     // is equivalent to the above
  ↦ // ** redefining j(3) (  ideal j(n);     // is equivalent to the above) .\
     /examples/Names_1.sing:4
    ideal j(2)=x;
    j(2..3);
  ↦ j(2)[1]=x j(3)[1]=0
    ring r=0,(x(1..2)(1..3)(1..2)),dp;
    r;
  ↦ // coefficients: QQ
  ↦ // number of vars : 12
  ↦ //        block   1 : ordering dp
  ↦ //                  : names    x(1)(1)(1) x(1)(1)(2) x(1)(2)(1) x(1)(2)(2\
    ) x(1)(3)(1) x(1)(3)(2) x(2)(1)(1) x(2)(1)(2) x(2)(2)(1) x(2)(2)(2) x(2)(\
    3)(1) x(2)(3)(2)
  ↦ //        block   2 : ordering C
    int i(1,2),i(2,3);
    i(2,3);
  ↦ 0
```

Names must not coincide with reserved names (keywords). Type `reservedName();` to get a list of the reserved names. See Section 5.1.135 [reservedName], page 254. Names should not interfere with names of ring variables or, more generally, with monomials. See Section 6.5 [Identifier resolution], page 315.

The command `listvar` provides a list of the names in use (see Section 5.1.82 [listvar], page 213).

The most recently printed expression is available under the special name `_`, e.g.,

```
    ring r;
    ideal i=x2+y3,y3+z4;
    std(i);
  ↦ _[1]=y3+x2
  ↦ _[2]=z4-x2
    ideal k=_;
    k*k+x;
  ↦ _[1]=y6+2x2y3+x4
  ↦ _[2]=y3z4+x2z4-x2y3-x4
  ↦ _[3]=z8-2x2z4+x4
  ↦ _[4]=x
    size(_[3]);
  ↦ 3
```

A string_expression enclosed in `‘...‘` (back ticks) evaluates to the value of the variable given by the string_expression. This feature is referred to as name substitution.

```
  int foo(1)=42;
  string bar="foo";
  ‘bar+"(1)"‘;
↦ 42
```

3.5.4 Objects

Every object in Singular has a type and a value. In most cases it has also a name and in some cases an attribute list. The value of an object may be examined simply by printing it with a print command: object;. The type of an object may be determined by means of the `typeof` function, the attributes by means of the `attrib` function (Section 5.1.161 [typeof], page 282, Section 5.1.2 [attrib], page 156):

```
  ring r=0,x,dp;
  typeof(10);
↦ int
  typeof(10000000000000000);
↦ bigint
  typeof(r);
↦ ring
  attrib(x);
↦ no attributes
  attrib(std(ideal(x)));
↦ attr:isSB, type int
```

Each object of type `poly`, `ideal`, `vector`, `module`, `map`, `matrix`, `number`, or `resolution` belongs to a specific ring. This is also true for `list`, if at least one of the objects contained in the list belongs to a ring. These objects are local to the ring. Their names can be duplicated for other objects in other rings. Objects from one ring can be mapped to another ring using maps or the commands `fetch` or `imap`. See Section 4.12 [map], page 106, Section 5.1.38 [fetch], page 182, Section 5.1.59 [imap], page 198.

All other types do not belong to a ring and can be accessed within every ring and across rings. They can be declared even if there is no active basering.

3.5.5 Type conversion and casting

Type conversion

Assignments convert the type of the right-hand side to the type of the left-hand side of the assignment, if possible. Operators and functions which require certain types of operands can also implicitly convert the type of an expression. It is, for example, possible to multiply a polynomial by an integer because the integer is automatically converted to a polynomial. Type conversions do not act transitively. Possible conversions are:

1. `intvec` ↦ `intmat`
2. `poly` ↦ `ideal`
3. `bigint` ↦ `ideal`
4. `int` ↦ `ideal`
5. `intmat` ↦ `matrix`
6. `ideal` ↦ `matrix`
7. `module` ↦ `matrix`
8. `number` ↦ `matrix`
9. `poly` ↦ `matrix`

10. vector ↦ matrix
11. bigint ↦ matrix
12. int ↦ matrix
13. intvec ↦ matrix
14. ideal ↦ module
15. matrix ↦ module
16. vector ↦ module
17. bigint ↦ number
18. int ↦ number
19. number ↦ poly
20. bigint ↦ poly
21. int ↦ poly
22. list ↦ resolution
23. poly ↦ vector (p ↦ p*gen(1))
24. bigint ↦ vector
25. int ↦ vector (i ↦ i*gen(1))
26. int ↦ bigint
27. int ↦ intvec
28. string ↦ link
29. resolution ↦ list
30. intvec ↦ bigintvec

Type casting

An expression can be casted to another type by using a type cast expression:
type (expression).

Possible type casts are:

to	from
bigint	expression int, number, poly
ideal	expression lists of int, number, poly
ideal	int, matrix, module, number, poly, vector
int	number, poly
intvec	expression lists of int, intmat, bigintvec
intmat	intvec (see Section 4.8.3 [intmat type cast], page 92)
list	expression lists of any type
matrix	module, ideal, vector, matrix. There are two forms to convert something to a matrix: if matrix(expression) is used then the size of the matrix is determined by the size of expression. But matrix(expression , m , n) may also be used - the result is a $m \times n$ matrix (see Section 4.13.3 [matrix type cast], page 110)
module	expression lists of int, number, poly, vector
module	ideal, matrix, vector
number	poly
poly	int, number
ring	list (the inverse of ringlist)
string	any type (see Section 4.22.3 [string type cast], page 131)

Example:

```
  ring r=0,x,(c,dp);
  number(3x);
↦ 0
```

```
  number(poly(3));
↦ 3
  ideal i=1,2,3,4,5,6;
  print(matrix(i));
↦ 1,2,3,4,5,6
  print(matrix(i,3,2));
↦ 1,2,
↦ 3,4,
↦ 5,6
  vector v=[1,2];
  print(matrix(v));
↦ 1,
↦ 2
  module(matrix(i,3,2));
↦ _[1]=[1,3,5]
↦ _[2]=[2,4,6]
  // generators are columns of a matrix
```

3.5.6 Flow control

A block is a sequence of commands surrounded by `{` and `}`.

```
{
   command;
   ...
}
```

Blocks are used whenever Singular is used as a structured programming language. The `if` and `else` structures allow conditional execution of blocks (see Section 5.2.9 [if], page 296, Section 5.2.5 [else], page 292). `for` and `while` loops are available for a repeated execution of blocks (see Section 5.2.8 [for], page 295, Section 5.2.15 [while], page 301). In procedure definitions, the main part and the example section are blocks as well(see Section 4.18 [proc], page 124).

3.6 Input and output

Singular's input and output (short, I/O) are realized using links. Links are the communication channels of Singular, i.e., something Singular can write to and read from. In this section, a short overview of the usage of links and of the different link types is given.

For loading of libraries, see Section 5.1.79 [LIB], page 211. For executing program scripts, see Section 5.1.41 [filecmd], page 184.

Monitoring

A special form of I/O is monitoring. When monitoring is enabled, Singular makes a typescript of everything printed on your terminal to a file. This is useful to create a protocol of a Singular session. The `monitor` command enables and disables this feature (see Section 5.1.95 [monitor], page 224).

How to use links

Recall that links are the communication channels of Singular, i.e., something Singular can write to and read from using the functions `write` and `read`. There are furthermore the functions `dump` and `getdump` which store resp. retrieve the content of an entire Singular session to, resp. from, a link. The `dump` and `getdump` commands are not available for DBM links.

For more information, see Section 5.1.174 [write], page 289, Section 5.1.130 [read], page 250, Section 5.1.27 [dump], page 175, Section 5.1.52 [getdump], page 191.

Example:

```
  ring r; poly p = x+y;
  dump(":w test.sv");   // dump the session to the file test.sv
  kill r;               // kill the basering
  listvar();            // no output after killing the ring
  getdump(":r test.sv");// read the dump from the file
  listvar();
↦ // r                              [0]  *ring
↦ //      p                              [0]  poly
```

Specifying a link can be as easy as specifying a filename as a string. Links do not even need to be explicitly opened or closed before, resp. after, they are used. To explicitly open or close a link, the `open`, resp. `close`, commands may be used (see Section 5.1.110 [open], page 234, Section 5.1.10 [close], page 163).

Links have various properties which can be queried using the `status` function (see Section 5.1.150 [status], page 270).

Example:

```
  link l = "ssi:fork";
  l;
↦ // type : ssi
↦ // mode : fork
↦ // name :
↦ // open : no
↦ // read : not open
↦ // write: not open
  open(l);
  status(l, "open");
↦ yes
  close(l);
  status(l, "open");
↦ no
```

ASCII links

Data that can be converted to a string can be written into files for storage or communication with other programs. The data are written in plain ASCII format. Reading from an ASCII link returns a string — conversion into other data is up to the user. This can be done, for example, using the command `execute` (see Section 5.1.32 [execute], page 178).

ASCII links should primarily be used for storing small amounts of data, especially if it might become necessary to manually inspect or manipulate the data.

See Section 4.10.4 [ASCII links], page 98, for more information.

Example:

```
  // (over)write file test.ascii, link is specified as string
  write(":w test.ascii", "int i =", 3, ";");
  // reading simply returns the string
  read("test.ascii");
↦ int i =
↦ 3
↦ ;
```

```
↦
  // but now test.ascii is "executed"
  execute(read("test.ascii"));
  i;
↦ 3
```

Ssi links

Data is communicated with other processes (e.g., Singular processes) which may run on the same computer or on different ones. Data exchange is accomplished using TCP/IP links in the ssi format. Reading from an ssi link returns the written expressions (i.e., not a string, in general).

Ssi links should primarily be used for communicating with other programs or for parallel computations (see, for example, Section A.1.8 [Parallelization with ssi links], page 707).

See Section 4.10.5 [Ssi links], page 99, for more information.

Example:

```
  ring r;
  link l = "ssi:tcp localhost:"+system("Singular"); // declare a link explicitly
  open(l);  // needs an open, launches another SINGULAR as a server
  write(l, x+y);
  kill r;
  def p = read(l);
  typeof(p); p;
↦ poly
↦ x+y
  close(l); // shuts down SINGULAR server
```

DBM links

Data is stored in and accessed from a data base. Writing is accomplished by a key and a value and associates the value with the key in the specified data base. Reading is accomplished w.r.t. a key, the value associated to it is returned. Both the key and the value have to be specified as strings. Hence, DBM links may be used only for data which may be converted to or from strings.

DBM links should primarily be used when data needs to be accessed not in a sequential way (like with files) but in an associative way (like with data bases).

See Section 4.10.7 [DBM links], page 102, for more information.

Example:

```
  ring r;
  // associate "x+y" with "mykey"
  write("DBM:w test.dbm", "mykey", string(x+y));
  // get from data base what is stored under "mykey"
  execute(read("DBM: test.dbm", "mykey"));
↦ x+y
```

3.7 Procedures

Procedures contain sequences of commands in the Singular language. They are used to extend the set of commands by user defined commands. In a Singular session, procedures are defined by either typing them on the command line or by loading them from a library file with the `LIB` or `load` command (see Section 3.8 [Libraries], page 55). A procedure is invoked like normal built-in commands, i.e., by typing its name followed by the list of arguments in parentheses. The invocation then executes the sequence of commands constituting the procedure. All procedures defined in a SINGULAR session can be displayed by entering `listvar(proc);` .

See also See Section 3.8.6 [Procedures in a library], page 57.

3.7.1 Procedure definition

Syntax: [static] proc proc_name [(<parameter_list>)]
[<help_string>]
{
  <procedure_body>
}
[example
{
  <sequence_of_commands>
}]

Purpose:

- Defines a new function, the proc proc_name.
- The help string, the parameter list, and the example section are optional. They are, however, mandatory for the procedures listed in the header of a library. The help string is ignored and no example section is allowed if the procedure is defined interactively, i.e., if it is not loaded from a file by the `LIB` or `load` command (see Section 5.1.79 [LIB], page 211 and see Section 5.2.12 [load], page 299).
- Once loaded from a file into a SINGULAR session, the information provided in the help string will be displayed upon entering `help proc_name;`, while the `example` section will be executed upon entering `example proc_name;`. See Section 3.7.2 [Parameter list], page 52, Section 3.7.3 [Help string], page 54, and the example in Section 3.8.6 [Procedures in a library], page 57.
- In the body of a library, each procedure not meant to be accessible by users should be declared static. See Section 3.8.6 [Procedures in a library], page 57.

Example of an interactive procedure definition and its execution:

```
  proc milnor_number (poly p)
  {
    ideal i= std(jacob(p));
    int m_nr=vdim(i);
    if (m_nr<0)
    {
      "// not an isolated singularity";
    }
    return(m_nr);         // the value of m_nr is returned
  }
  ring r1=0,(x,y,z),ds;
  poly p=x^2+y^2+z^5;
  milnor_number(p);
↦ 4
```

Example of a procedure definition in a library:

First, we define the library (and store it as `sample.lib`):

```
// Example of a user accessible procedure
proc tab (int n)
"USAGE:    tab(n);  n integer
RETURNS:  string of n space tabs
EXAMPLE:  example tab; shows an example"
{ return(internal_tab(n)); }
example
{
  "EXAMPLE:"; echo=2;
  for(int n=0; n<=4; n=n+1)
  { tab(4-n)+"*"+tab(n)+"+"+tab(n)+"*"; }
}

// Example of a static procedure
static proc internal_tab (int n)
{ return(" "[1,n]); }
```

Now, we load the library and execute its procedures:

```
  LIB "sample.lib";        // load the library sample.lib
  example tab;             // show an example
↦ // proc tab from lib sample.lib
↦ EXAMPLE:
↦   for(int n=0; n<=4; n=n+1)
↦   { tab(4-n)+"*"+tab(n)+"+"+tab(n)+"*"; }
↦     *+*
↦    * + *
↦   *  +  *
↦  *   +   *
↦ *    +    *
↦ 
  "*"+tab(3)+"*";          // use the procedure tab
↦ *   *
  // the static procedure internal_tab is not accessible
  "*"+internal_tab(3)+"*";
↦    ? `internal_tab(3)` is not defined
↦    ? error occurred in or before ./examples/Example_of_a_procedure_defini\
   tion_in_a_library:.sing line 5: `  "*"+internal_tab(3)+"*";`
  // show the help section for tab
  help tab;
↦ // ** Could not get 'IdxFile'.
↦ // ** Either set environment variable 'SINGULAR_IDX_FILE' to 'IdxFile',
↦ // ** or make sure that 'IdxFile' is at "%D/singular/singular.idx"
↦ // ** Displaying help in browser 'dummy'.
↦ // ** Use 'system("--browser", <browser>);' to change browser,
↦ // ** where <browser> can be: "dummy", "emacs".
↦    ? No functioning help browser available.
↦    ? error occurred in or before ./examples/Example_of_a_procedure_defini\
   tion_in_a_library:.sing line 7: `  help tab;`
```

3.7.2 Parameter list

Syntax: ( )
(parameter_definition)

Purpose:

- Defines the number, type and names of the arguments of a procedure.
- The parameter_list is optional.
- Adding `list #` as argument to a parameter list means to allow optional parameters. Furthermore, `(list #)` is the default for a parameter list (in case no list is explicitly given). Inside the procedure body, the arguments of `list #` are referenced by `#[1]`, `#[2]`, etc.
- If a procedure has optional parameters, the attribute `default_arg` gives the default values for the optional arguments. This provides in particular the possibility to also change the behaviour of all procedures nested inside the given procedure.

Example:

```
proc x0
{
    // can be called with
... // any number of arguments of any type: #[1], #[2],...
    // number of arguments: size(#)
}

proc x1 ()
{
... // can only be called without arguments
}

proc x2 (ideal i, int j)
{
... // can only be called with 2 arguments,
    // which can be converted to ideal resp. int
}

proc x3 (i,j)
{
... // can only be called with 2 arguments
    // of any type
    // (i,j) is the same as (def i,def j)
}

proc x5 (i,list #)
{
... // can only be called with at least 1 argument
    // number of arguments: size(#)+1
}

attrib(x5,"default_arg",3);
x5(2); // is equivalent to
x5(2,3);
```

Note:

The parameter_list may stretch across multiple lines.
A parameter may have any type (including the types `proc` and `ring`).
If a parameter is of type ring, then it can only be specified by name, but not with a type. For instance:

```
proc x6 (r)
{
... // this is correct, r may be of any type, even of type ring
}

proc x7 (ring r)
{
... // this is NOT CORRECT
}
```

3.7.3 Help string

Syntax: string_constant;

Purpose: Constitutes the help text of a procedure.

Format:

```
USAGE:    <proc_name>(<parameter list>);   <explanation of parameters>
ASSUME:  <description of assumptions made>
RETURN:   <description of what is returned>
SIDE EFFECTS:  <description of global objects generated or manipulated,
but not returned>
REMARKS: <information on theory and implemented  algorithms,references>
NOTE:    <particularities, limitations, additional details>
KEYWORDS: <semicolon-separated phrases of index keys>
SEE ALSO: <comma-separated names of related procedures/cross references>
EXAMPLE:  example <proc_name>; shows an example
```

NOTE:

- ASSUME, SIDE EFFECTS, KEYWORDS, and SEE ALSO are optional. No help string is required for static procedures.
- EXAMPLE: refers to the example section of the procedure. In a Singular session, the example will be carried out upon entering `example <proc_name>;` if the procedure is loaded from a file by the `LIB` or `load` command (see Section 5.1.79 [LIB], page 211 and see Section 5.2.12 [load], page 299). No example section is allowed if the procedure is defined interactively.
- See Section 3.8.10 [Typesetting of help and info strings], page 64 for help strings in the Singular documentation.
- See the example in Section 3.8.6 [Procedures in a library], page 57 for an illustration.

3.7.4 Names in procedures

- All variables defined inside a procedure are local to the procedure and their names cannot interfere with names in other procedures. Without further action, they are automatically deleted after leaving the procedure.
- To keep local variables and their value after leaving the procedure, they have to be exported (i.e. made global) by a command like `export` or `exportto` (see Section 5.2.6 [export], page 292,see Section 5.2.7 [exportto], page 293, see Section 5.2.10 [importfrom], page 296; see Section 4.16 [package], page 119). To return the value of a local variable, use the `return` command (see Section 5.2.14 [return], page 301).

Example:

```
proc xxx
{
  int k=4;        //defines a local variable k
  int result=k+2;
  export(result);  //defines the global variable "result".
}
xxx();
listvar(all);
↦ // result                          [0]  int 6
```

Note that the variable `result` became a global variable after the execution of `xxx`.

3.7.5 Procedure-specific commands

A few commands should only be used inside a procedure. They either make local objects global ones or return results to the level from where the procedure was called.

See Section 5.2.6 [export], page 292; Section 5.2.7 [exportto], page 293; Section 5.2.14 [return], page 301.

3.8 Libraries

- A library is a collection of Singular procedures in a file.
- To load a library into a Singular session, use the `LIB` or `load` command. Having loaded a library, its procedures can be used like any built-in Singular function, and information on the library is obtained by entering `help libname.lib;`
- See Appendix D [SINGULAR libraries], page 793, for all libraries currently distributed with Singular.
- When writing your own library, it is important to comply with the guidelines described in this section. Otherwise, due to potential parser errors, it may not be possible to load the library.
- Each library consists of a header and a body. The first line of a library must start with a double slash `//`.
- The library header consists of a version string, a category string, an info string, and LIB commands. The strings are mandatory. LIB commands are meant to load the additional libraries used by the library under consideration.
- The library body collects the procedures (declared static or not).
- No line of a library should consist of more than 60 characters.

3.8.1 Libraries in the SINGULAR Documentation

- The typesetting language in which the Singular documentation is written is `texinfo`. The info string of a library included in the Singular distribution will be parsed and automatically translated to the `texinfo` format. The same applies to the help string of each procedure listed in the PROCEDURE: section of the info string.
- Based on various tools, `info`, `dvi`, `p`, and `html` versions of the `texinfo` documentation are generated.
- For texinfo markup elements and other information facilitating optimal typesetting, see Section 3.8.10 [Typesetting of help and info strings], page 64.
- For the convenience of users checking directly the source code, the `texinfo` tools should be used economically. That is, the info and help texts should be well readable verbatim.
- The example of each procedure listed in the PROCEDURE: section of the info string is computed and its output is included in the documentation.

3.8.2 Version string

A version string is part of the header of a library.

Syntax: version = string_constant;

Purpose: Defines the version number of a library. It is displayed when the library is loaded.

Example: `version="version sample.lib 4.0.0.0 Dec_2013 ";`

Note: Syntax: `version<space><filename><space><version><space><date><space>`

3.8.3 Category string

A category string is part of the header of a library.

Syntax: category = string_constant;

Purpose: Defines the category of a library.

Example: category="Algebraic geometry";

Note: Reserved for sorting the libraries into categories.

3.8.4 Info string

Syntax: info = string_constant;

Purpose: Constitutes the help text of a library. Will be displayed in a Singular session upon entering `help libname.lib;` . Will be part of the Singular documentation if the library is distributed with Singular. See Section 3.8.1 [Libraries in the SINGULAR Documentation], page 55.

Format:

```
info="
LIBRARY: <library_name> <one line description of the purpose>
AUTHOR:  <name, and email address of author>
OVERVIEW: <concise, additional information on what is implemented>
REFERENCES: <references for further information>
KEYWORDS: <semicolon-separated phrases of index keys>
SEE ALSO: <comma-separated words of cross references>
PROCEDURES:
  <proc_name_1>();     <one line description of the purpose>
       .
       .
  <proc_name_N>();     <one line description of the purpose>
";
```

NOTE:

- In the documentation, the one line description of the purpose following LIBRARY: will be printed in its own line, starting with the prefix PURPOSE: .
- REFERENCES, KEYWORDS, and SEE ALSO are optional.
- Only non-static procedures should be listed in the PROCEDURES: section. A procedure parameter should be included between the brackets `()` only if the corresponding one line description of the purpose refers to it. See Section 3.8.6 [Procedures in a library], page 57.

- In the documentation, separate nodes (subsections in printed documents) are created precisely for those procedures of the library appearing n the PROCEDURES: section (that is, for some if not all non-static procedures of the library).

Example:

```
info="
LIBRARY: absfact.lib   Absolute factorization for characteristic 0
AUTHORS: Wolfram Decker,       decker at math.uni-sb.de
         Gregoire Lecerf,      lecerf at math.uvsq.fr
         Gerhard Pfister,      pfister at mathematik.uni-kl.de

OVERVIEW:
A library for computing the absolute factorization of multivariate
polynomials f with coefficients in a field K of characteristic zero.
Using Trager's idea, the implemented algorithm computes an absolutely
irreducible factor by factorizing over some finite extension field L
(which is chosen such that V(f) has a smooth point with coordinates in L).
Then a minimal extension field is determined making use of the
Rothstein-Trager partial fraction decomposition algorithm.

REFERENCES:
G. Cheze, G. Lecerf: Lifting and recombination techniques for absolute
                  factorization. Journal of Complexity, 23(3):380-420, 2007

KEYWORDS: factorization; absolute factorization.
SEE ALSO: factorize

PROCEDURES:
  absFactorize();        absolute factorization of poly
";
```

To see how this infostring appears in the documentation after typesetting, check Section D.4.1 [absfact_lib], page 1001:

3.8.5 LIB commands

LIB commands are part of the header of a library.

Syntax: LIB "lib_1.lib";
...
LIB "lib_r.lib";

Purpose: Loads libraries used by the library under consideration.

Example:

```
LIB "primdec.lib";
LIB "normal.lib";
```

Note: The keyword LIB must be followed by at least one space.

3.8.6 Procedures in a library

Here asre hints and requirements on how procedures contained in a library should be implemented. For more on procedures, see Section 3.7 [Procedures], page 50.

1. Each procedure not meant to be accessible by users should be declared static.

2. The header of each procedure not declared static must comply with the guidelines described in Section 3.7.1 [Procedure definition], page 51 and Section 3.7.3 [Help string], page 54. In particular, it must have a help and example section, and assumptions made should be carefully explained. If the assumptions are checked by the procedure on run-time, errors may be reported using the Section 5.1.30 [ERROR], page 177 function.
3. Names of procedures should not be shorter than 4 characters and should not contain any special characters. In particular, the use of _ in names of procedures is discouraged. If the name of the procedure is composed of more than one word, each new word should start with a capital letter, all other letters should be lower case (e.g. linearMapKernel).
4. No procedures should be defined within the body of another procedure.
5. A procedure may print out comments, for instance to explain results or to display intermediate computations. This is often helpful when calling the procedure directly, but it may also cause confusions in cases where the procedure is called by another procedure. The SINGULAR solution to this problem makes use of the function `dbprint` (see Section 5.1.17 [dbprint], page 169) and the reserved variables `printlevel` and `voice` (see Section 5.3.6 [printlevel], page 304 and see Section 5.3.11 [voice], page 308). Note that `printlevel` is a predefined, global variable whose value can be changed by the user, while `voice` is an internal variable, representing the nesting level of procedures. Accordingly, the value of Section 5.3.11 [voice], page 308 is 1 on the top level, 2 inside the first procedure, and so on. The default value of `printlevel` is 0, but `printlevel` can be set to any integer value by the user.

Example: If the procedure `Test` below is called directly from the top level, then ‘comment1’ is displayed, but not ‘comment2’. By default, nothing is displayed if `Test` is called from within any other procedure. However, if `printlevel` is set to a value k with k>0, then ‘comment1’ (resp. ‘comment2’) is displayed – provided `Test` is called from another procedure with nesting level at most k (resp. k-1).

The example part of a procedure behaves in this respect like the procedure on top level (the nesting level is 1, that is, the value of `voice` is 2). Therefore, due to the command `printlevel=1;`, ‘comment1’ will be displayed when entering `example Test;`. However, since printlevel is a global variable, it should be reset to its old value at the end of the example part.

The predefined variable `echo` controls whether input lines are echoed or not. Its default is 0, but it can be reset by the user. Input is echoed if `echo>=voice`. At the beginning of the example part, `echo` is set to the value 2. In this way, the input lines of the example will be displayed when entering `example Test;`.

```
proc Test
"USAGE:   ...
         ...
EXAMPLE: example Test; shows an example
"
{   ...
   int p = printlevel - voice + 3;
    ...
   dbprint(p,"comment1");
   dbprint(p-1,"comment2");
   // dbprint prints only if p > 0
    ...
}
example
{ "EXAMPLE:"; echo = 2;
   int p = printlevel;   //store old value of printlevel
```

```
                            printlevel = 1;       //assign new value to printlevel
                             ...
                            Test();
                            printlevel = p;       //reset printlevel to old value
                         }
```

Note: Singular functions such as `pause` or `read` allow and require interactive user-input. They are, thus, in particular useful for debugging purposes. If such a command is used inside the procedure of a library to be distributed with Singular, the example section of the procedure has to be written with some care – the procedure should only be called from within the example if the value of `printlevel` is 0. Otherwise, the automatic build process of Singular will not run through since the examples are carried out during the build process. They are, thus, tested against changes in the code.

3.8.7 template_lib

First, we show the source-code of a template library:

```
////////////////////////////////////////////////////////////////////
version="version template.lib 4.1.2.0 Feb_2019 "; // $Id: 4d4a314bcbeaaaf113c4c4687bb
category="Miscellaneous";
// summary description of the library
info="
LIBRARY:   template.lib  A Template for a Singular Library
AUTHOR:    Olaf Bachmann, email: obachman@mathematik.uni-kl.de

SEE ALSO:  standard_lib, Libraries,
           Typesetting of help and info strings

KEYWORDS: library, template.lib; template.lib; library, info string

PROCEDURES:
  mdouble(int)           return double of int argument
  mtriple(int)          return three times int argument
  msum([int,..,int])     sum of int arguments
";
////////////////////////////////////////////////////////////////////
proc mdouble(int i)
"USAGE:    mdouble(i); i int
RETURN:   int: i+i
NOTE:     Help string is in pure ASCII.
          This line starts on a new line since previous line is short.
          No new line here.
SEE ALSO: msum, mtriple, Typesetting of help and info strings
KEYWORDS: procedure, ASCII help
EXAMPLE:  example mdouble; shows an example"
{
  return (i + i);
}
example
{ "EXAMPLE:"; echo = 2;
  mdouble(0);
  mdouble(-1);
}
```

```
////////////////////////////////////////////////////////////////////
proc mtriple(int i)
"@c we do texinfo here
@table @asis
@item @strong{Usage:}
@code{mtriple(i)}; @code{i} int

@item @strong{Return:}
int: @math{i+i+i}
@item @strong{Note:}
Help is in pure Texinfo.
@*This help string is written in texinfo, which enables you to use,
among others, the @@math command for mathematical typesetting
(for instance, to print @math{\alpha, \beta}).
@*Texinfo also gives more control over the layout, but is, admittingly,
more cumbersome to write.
@end table
@c use @c ref contstuct for references
@cindex procedure, texinfo help
@c ref
@strong{See also:}
@ref{mdouble}, @ref{msum}, @ref{Typesetting of help and info strings}
@c ref
"
{
  return (i + i + i);
}
example
{ "EXAMPLE:"; echo = 2;
  mtriple(0);
  mtriple(-1);
}
////////////////////////////////////////////////////////////////////
proc msum(list #)
"USAGE:  msum([i_1,..,i_n]); @code{i_1,..,i_n} def
RETURN:  Sum of int arguments
NOTE:    This help string is written in a mixture of ASCII and texinfo.
         @* Use @ref for references (e.g.,  @pxref{mtriple}).
         @* Use @code for typewriter font (e.g., @code{i_1}).
         @* Use @math for simple math mode typesetting (e.g., @math{i_1}).
         @* Warning: Parenthesis like } are not allowed inside @math and @code.
         @* Use @example for indented, preformatted text typesetting in
         typewriter font:
@example
         this  --> that
@end example
        Use @format for preformatted text typesetting in normal font:
@format
         this --> that
@end format
        Use @texinfo for text in pure texinfo:
@texinfo
@expansion{}
```

```
@tex
$i_{1,1}$
@end tex

@end texinfo
        Note that
        automatic linebreaking         is still in affect (like in this line).
SEE ALSO: mdouble, mtriple, Typesetting of help and info strings
KEYWORDS: procedure, ASCII/Texinfo help
EXAMPLE: example msum; shows an example"
{
  if (size(#) == 0) { return (0);}
  if (size(#) == 1) { return (#[1]);}
  int i;
  def s = #[1];
  for (i=2; i<=size(#); i++)
  {
    s = s + #[i];
  }
  return (s);
}
example
{ "EXAMPLE:"; echo = 2;
  msum();
  msum(4);
  msum(1,2,3,4);
}
```

Second, we show how the library appears in the documentation after typesetting (with one subsection for each procedure):

Library: template.lib

Purpose: A Template for a Singular Library

Author: Olaf Bachmann, email: obachman@mathematik.uni-kl.de

Procedures: See also: Section 3.8 [Libraries], page 55; Section 3.8.10 [Typesetting of help and info strings], page 64; Section D.1 [standard_lib], page 793.

3.8.7.1 mdouble

Procedure from library `template.lib` (see Section 3.8.7 [template_lib], page 59).

Usage: mdouble(i); i int

Return: int: i+i

Note: Help string is in pure ASCII.
This line starts on a new line since previous line is short. No new line here.

Example:

```
LIB "template.lib";
mdouble(0);
↦ 0
mdouble(-1);
↦ -2
```

See also: Section 3.8.10 [Typesetting of help and info strings], page 64; Section 3.8.7.3 [msum], page 62; Section 3.8.7.2 [mtriple], page 62.

3.8.7.2 mtriple

Procedure from library `template.lib` (see Section 3.8.7 [template_lib], page 59).

Usage: `mtriple(i)`; `i` int

Return: int: $i + i + i$

Note: Help is in pure Texinfo.
This help string is written in texinfo, which enables you to use, among others, the `@math` command for mathematical typesetting (for instance, to print α, β).
Texinfo also gives more control over the layout, but is, admittingly, more cumbersome to write.

See also:

Example:

```
LIB "template.lib";
mtriple(0);
↦ 0
mtriple(-1);
↦ -3
```

3.8.7.3 msum

Procedure from library `template.lib` (see Section 3.8.7 [template_lib], page 59).

Usage: msum([i_1,..,i_n]); `i_1,..,i_n` def

Return: Sum of int arguments

Note: This help string is written in a mixture of ASCII and texinfo.
Use `@ref` for references (e.g., see Section 3.8.7.2 [mtriple], page 62).
Use `@code` for typewriter font (e.g., `i_1`).
Use `@math` for simple math mode typesetting (e.g., i_1).
Warning: Parenthesis like `}` are not allowed inside `@math` and `@code`.
Use `@example` for indented, preformatted text typesetting in typewriter font:

```
          this  --> that
```

Use `@format` for preformatted text typesetting in normal font:

this –> that

Use `@texinfo` for text in pure texinfo:

↦ $i_{1,1}$

Note that
automatic linebreaking is still in affect (like in this line).

Example:

```
LIB "template.lib";
msum();
↦ 0
msum(4);
↦ 4
msum(1,2,3,4);
```

```
↦ 10
```

See also: Section 3.8.10 [Typesetting of help and info strings], page 64; Section 3.8.7.1 [mdouble], page 61; Section 3.8.7.2 [mtriple], page 62.

3.8.8 Formal Checker

There is a formal library checker for SINGULAR which can be used online: see `https://www.singular.uni-kl.de/index.php/new-libraries/formal-library-checker.html`.

After uploading your library file, you will receive an output of hints, warnings, and errors which may help you to improve your library.

3.8.9 Documentation Tool

`lib2doc` is a utility to generate the stand-alone documentation for a Singular library in various formats.

The `lib2doc` utility should be used by developers of Singular libraries to check the generation of the documentation of their libraries.

`lib2doc` can be downloaded from
`ftp://jim.mathematik.uni-kl.de/pub/Math/Singular/misc/lib2doc.tar.gz` (mirror at `https://www.mathematik.uni-kl.de/ftp/pub/Math/Singular/misc/lib2doc.tar.gz`)

Important:
To use `lib2doc`, you need to have `perl` (version 5 or higher), `texinfo` (version 3.12 or higher) and `Singular` and `libparse` (version 1-3-4 or higher) installed on your system.

To generate the documentation for a library, follow these steps:

1. Unpack lib2doc.tar.gz

   ```
   gzip -dc  lib2doc.tar.gz | tar -pxf -
   ```

   and

   ```
   cd lib2doc
   ```

2. Edit the beginning of the file `Makefile`, filling in the values for `SINGULAR` and `LIBPARSE`. Check also the values of `PERL` and `LATEX2HTML`.
3. Copy your library to the current directory:

   ```
   cp <path-where-your-lib-is>/mylib.lib .
   ```

4. Now you can run the following commands:

   `make mylib.hlp`
   Generates the file `mylib.hlp` – the info file for the documentation of `mylib.lib`. This file can be viewed using

   ```
   info -f mylib.hlp
   ```

   `make mylib.dvi`
   Generates the file `mylib.dvi` – the dvi file for the documentation of `mylib.lib`. This file can be viewed using

   ```
   xdvi mylib.dvi
   ```

   `make mylib.ps`
   Generates the file `mylib.ps` – the PostScript file for the documentation of `mylib.lib`. This file can be viewed using (for example)

   ```
   ghostview mylib.dvi
   ```

`make mylib.html`
: Generates the file `mylib.html` – the HTML file for the documentation of `mylib.lib`. This file can be viewed using (for example)

```
firefox mylib.html
```

`make clean`
: Deletes all generated files.

Note that you can safely ignore messages complaining about undefined references.

3.8.10 Typesetting of help and info strings

The info strings of the libraries which are included in the distribution of SINGULAR and the help strings of the corresponding procedures are parsed and automatically converted into the `texinfo` format (the typesetting language in which the documentation of SINGULAR is written).

The illustrative example given in Section 3.8.7 [template_lib], page 59 should provide sufficient information on how this works. For more details, check the following items:

- Users familiar with `texinfo` may write help and info strings directly in the `texinfo` format. The string should, then, start with the `@` sign. In this case, no parsing will be done.
- Help and info strings are typeset within a `@table @asis` environment (which is similar to the `latex description` environment).
- If a line starts with uppercase words up to a colon, then the text up to the colon is taken to be the description-string of an item, and the text following the colon is taken to be the content of the item.
- If the description-string of an item matches

  `SEE ALSO` then the content of the item is assumed to consist of comma-separated words which are valid references to other `texinfo` nodes of the manual (e.g., all procedure and command names are also `texinfo` nodes).

  `KEYWORDS` then the content of the item is assumed to be a semicolon-separated list of phrases which are taken as keys for the index of the manual (the name of a procedure/library is automatically added to the index keys).
- If the description-string of an item in the **info string of a library** matches

  `LIBRARY` then the content of the item is assumed to be a one-line description of the library. If this one-line description consists of uppercase characters only, then it is typeset in lowercase characters (otherwise it is left as is).

  `PROCEDURES`
  then the content of the item is assumed to consist of lines of type

  ```
  <proc_name>();      <one line description of the purpose>
  ```

  Separate `texinfo` nodes (subsections in printed documents) are created precisely for those procedures of the library appearing here (that is, for some if not all non-static procedures of the library).

With respect to the content of an item, the following `texinfo` markup elements are recognized:

`@*` Enforces a line-break.

Example: `old line @* new line`
↦
old line
new line

`@ref{...}` For references to other parts of the Singular manual, use one of the following `@ref{node}` constructs. Here, `node` must be the name of a section of the Singular manual. In particular, it may be the name of a function, library or procedure in a library.

`@xref{node}`
: for a reference to the node `node` at the beginning of a sentence.

`@ref{node}`
: for a reference to the node `node` at the end of a sentence.

`@pxref{node}`
: for a reference to the node `node` within parentheses.

Example: `@xref{Hurricanes}, for more info.`
↦*Note Hurricanes::, for more info.
↦See Section 3.1 [Hurricanes], page 24, for more info.

`For more information, see @ref{Hurricanes}.`
↦For more information, see *Note Hurricanes::.
↦For more information, see Section 3.1 [Hurricanes], page 24.

`... storms cause flooding (@pxref{Hurricanes}) ...`
↦... storms cause flooding (*Note Hurricanes::) ...
↦... storms cause flooding (see Section 3.1 [Hurricanes], page 24)

`@math{..}` Typeset short mathematical expressions in LaTeX math-mode syntax (short: does not cause expansion over multiple lines).

Example: `@math{\alpha}`
↦
α

Note: The mathematical expressions inside `@math{..}` must not contain the characters `{`,`}`, and `@`.

`@code{..}` Typeset short strings in typewriter font (short: does not cause expansion over multiple lines).

Example: `@code{typewriter font}`
↦
`typewriter font`

Note: The string inside `@code{..}` must not contain the characters `{`,`}`, and `@`.

Typeset pre-formatted text in typewriter font.

```
@example
 ...
@end example
```

Example:

```
before example
@example
in example
notice escape of special characters like @{,@},@@
@end example
after example
```

↦
before example

```
          in example
          notice escape of special characters like {,},@
```

after example

Note: Inside an @example environment, the characters {,},@ have to be escaped by an @ sign.

Typeset pre-formatted text in normal font.

```
@format
 ...
@end format
```

Example:

```
          before format
          @format
          in format
          notice escape of special characters like @{,@},@@
          @end format
          after format
```

↦

before format

in format
escape of special characters like {,},@

after format

Note: Inside an @format environment, the characters {,},@ have to be escaped by an @ sign.

Write text in pure `texinfo`.

```
@texinfo
 ...
@end texinfo
```

Example:

```
          @texinfo
          Among others, within a texinfo environment,
          one can use the tex environment to typeset
          more complex mathematical items like
          @tex
          $i_{1,1} $
          @tex
          @end texinfo
```

↦

Among others, within a texinfo environment, one can use the tex environment to typeset more complex mathematical items like $i_{1,1}$

Furthermore, a line-break is inserted before each line whose previous line is shorter than 60 characters and does not contain any of the above described recognized texinfo markup elements.

3.8.11 Loading a library

Libraries can be loaded with the `LIB` or the `load` command (see Section 5.1.79 [LIB], page 211 and see Section 5.2.12 [load], page 299).

Syntax: `LIB` string_expression ;
`load` string_expression ;

Type: none

Purpose: Reads a library from a file. If the given filename does not start with `.` or `/` and if the file cannot be located in the current directory, the `SearchPath` is checked for a directory containing a file with this name.

Note on SearchPath:
The `SearchPath` for a library is constructed at Singular start-up time as follows:

1. the directories contained in the environment variable `SINGULARPATH` are appended.
2. the directories `$BinDir/LIB`, `$RootDir/LIB`, `$RootDir/../LIB`, `$DefaultDir/LIB`, `$DefaultDir/../LIB` are appended, where
   - `$BinDir` is the value of the environment variable `SINGULAR_BIN_DIR`, if set, or, if not set, the directory in which the Singular program resides
   - `$RootDir` is the value of the environment variable `SINGULAR_ROOT_DIR`, if set, or, if not set, `$BinDir/../`.
   - `$DefaultDir` is the value of the environment variable `SINGULAR_DEFAULT_DIR`, if set, or `/usr/local`.
3. all directories which do not exist are removed from the `SearchPath`.

For setting environment variables, see Section 5.1.155 [system], page 275, or consult the manual of your shell.

The library `SearchPath` can be examined by starting up Singular with the option `-v`, or by issuing the command `system("--version");`.

Note on standard.lib:
Unless Singular is started with the `--no-stdlib` option, the library `standard.lib` is automatically loaded at start-up time.

Following a `LIB` or `load` command, only the names of the procedures in the library are loaded. The body of a particular procedure is only read upon the first call of the procedure. This minimizes memory consumption by unused procedures. Starting a Singular session with the `-q` or `--quiet option` unsets the `option loadLib` and inhibits, thus, the monitoring of library loading (see `option`).

All libraries loaded in a Singular session are displayed upon entering `listvar(package);` :

```
  option(loadLib);    // show loading of libraries;
                      // standard.lib is loaded
  listvar(package);
↦ // Singmathic                  [0]  package Singmathic (C,singmathic.s\
  o)
↦ // Standard                    [0]  package Standard (S,standard.lib)
↦ // Top                         [0]  package Top (T)
                      // the names of the procedures of inout.lib
  LIB "inout.lib";    // are now known to Singular
↦ // ** loaded inout.lib (4.1.2.0,Feb_2019)
  listvar(package);
↦ // Inout                       [0]  package Inout (S,inout.lib)
↦ // Singmathic                  [0]  package Singmathic (C,singmathic.s\
  o)
↦ // Standard                    [0]  package Standard (S,standard.lib)
↦ // Top                         [0]  package Top (T)
```

See Section 3.1.6 [Command line options], page 19; Section 5.1.79 [LIB], page 211; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 793; Section 4.18 [proc], page 124; Section D.1 [standard_lib], page 793; Section 4.22 [string], page 130; Section 5.1.155 [system], page 275.

3.9 Debugging tools

If Singular does not come back to the prompt while calling a user defined procedure, probably a bracket or a " is missing. The easiest way to leave the procedure is to type some brackets or " and then ⟨RETURN⟩ .

3.9.1 ASSUME

Syntax: `ASSUME` (int_constant , expression)

Purpose: Tests the expression for correctness if the int_constant is smaller as a variable `assumeLevel`. If no such variable exist the int expression is compared against 0. It is possible to define an individual `assumeLevel` for each library and/or procedure If the expression is evaluated and not true (i.e. does not evaluate to `int(0)` an error is raised.

Note: `ASSUME shall be used for documentation and debugging,`
production code of a library must never define `assumeLevel`.

Example:

```
  ASSUME(0,2==2); // always tested
  ASSUME(1,1==2); // not evaluated
  int assumeLevel=2;
  ASSUME(1,1==2);
↦    ? ASSUME failed:  ASSUME(1,1==2);
↦    ? error occurred in or before ./examples/ASSUME.sing line 4: ‘   ASSUM
   (1,1==2);‘
  // setting a different assumeLevel for poly.lib:
  int Poly::assumeLevel=2;
↦ Poly of type ’ANY’. Trying load.
↦    ? ’Poly’ no such package
↦    ? error occurred in or before ./examples/ASSUME.sing line 6: ‘   int P
   ly::assumeLevel=2;‘
↦    ? wrong type declaration. type ’help int;’
```

3.9.2 Tracing of procedures

Setting the `TRACE` variable to 1 (resp. 3) results in reporting of all procedure entries and exits (resp. together with line numbers). If `TRACE` is set to 4, `Singular` displays each line before its interpretation and waits for the ⟨RETURN⟩ key being pressed. See Section 5.3.9 [TRACE var], page 306.

Example:

```
proc t1
{
  int i=2;
  while (i>0)
  { i=i-1; }
}
```

```
    TRACE=3;
    t1();
  ↦
  ↦ entering t1 (level 0)
  ↦ {1}{2}{3}{4}{5}{4}{5}{6}{7}{4}{5}{6}{7}{4}{6}{7}{8}
  ↦ leaving  t1 (level 0)
```

3.9.3 Source code debugger

The source code debugger (sdb) is an experimental feature, its interface may change in future versions of SINGULAR.
To enable the use of the source code debugger SINGULAR has to be started with the option `-d` or `--sdb` (see Section 3.1.6 [Command line options], page 19).

sdb commands

Each sdb command consists of one character which may be followed by a parameter.

`b` print backtrace of calling stack

`c` continue

`e` edit the current procedure and reload it (current call will be aborted)
only available on UNIX systems

`h,?` display help screen

`n` execute current line, sdb break at next line

`p` <identifier>
display type and value of the variable given by <identifier>

`Q` quit this SINGULAR session

`q` <flags> quit debugger, set debugger flags(0,1,2)
0: continue, disable the debugger
1: continue
2: throw an error, return to toplevel

Syntactical errors in procedures

If SINGULAR was started using the command line option `-d` or `--sdb`, a syntactical error in a procedure will start the source code debugger instead of returning to the top level with an error message. The commands `q 1` and `q 2` are equivalent in this case.

SDB breakpoints in procedures

Up to seven SDB breakpoints can be set. To set a breakpoint at a procedure use `breakpoint`. (See Section 5.2.3 [breakpoint], page 291).
These breakpoints can be cleared with the command `d breakpoint_no` from within the debugger or with `breakpoint(` proc_name `,-1);`.

3.9.4 Break points

A break point can be put into a proc by inserting the command `~`. If `Singular` reaches a break point it asks for lines of commands (line-length must be less than 80 characters) from the user. It returns to normal execution if given an empty line. See Section 5.2.16 [~], page 302.

Example:

```
proc t
{
  int i=2;
  ~;
  return(i+1);
}
t();
↦ -- break point in t --
↦ -- 0: called    from STDIN --
i;               // here local variables of the procedure can be accessed
↦ 2
↦ -- break point in t --

↦ 3
```

3.9.5 Printing of data

The procedure `dbprint` is useful for optional output of data: it takes 2 arguments and prints the second argument, if the first argument is positive; otherwise, it does nothing. See Section 5.1.17 [dbprint], page 169; Section 5.3.11 [voice], page 308.

3.9.6 libparse

`libparse` is a stand-alone program contained in the Singular distribution (at the place where the Singular executable program resides), which cannot be called inside Singular. It is a debugging tool for libraries which performs exactly the same checks as the `load` command in Singular, but generates more output during parsing. `libparse` is useful if an error occurs while loading the library, but the whole block around the line specified seems to be correct. In these situations the real error might have occurred hundreds of lines earlier in the library.

Usage:
`libparse [options] singular-library`
Options:

`-d` Debuglevel
: increases the amount of output during parsing, where Debuglevel is an integer between 0 and 4. Default is 0.

`-s` turns on reporting about violations of unenforced syntax rules

The following syntax checks are performed in any case:

- counting of pairs of brackets `{,}` , [,] and (,) (number of `{` has to match number of `}`, same for [,] and (,)).
- counting of `"` (number of `"` must be even).
- general library syntax (only LIB, static, proc (with parameters, help, body and example) and comments, i.e // and `/* ... */`, are allowed).

Its output lists all procedures that have been parsed successfully:

```
$ libparse sample.lib
Checking library 'sample.lib'
  Library         function      line,start-eod line,body-eob  line,example-eoe
Version:0.0.0;
g Sample               tab line    9,  149-165   13,  271-298   14,  300-402
l Sample      internal_tab line   24,  450-475   25,  476-496    0,    0-496
```

where the following abbreviations are used:

- g: global procedure (default)
- l: static procedure, i.e., local to the library.

each of the following is the position of the byte in the library.

- start: begin of 'proc'
- eod: end of parameters
- body: start of procedurebody '{'
- eob: end of procedurebody '}'
- example: position of 'example'
- eoe: end of example '}'

Hence in the above example, the first procedure of the library sample.lib is user-accessible and its name is tab. The procedure starts in line 9, at character 149. The head of the procedure ends at character 165, the body starts in line 13 at character 271 and ends at character 298. The example section extends from line 14 character 300 to character 402.

The following example shows the result of a missing close-bracket } in line 26 of the library `sample.lib`.

```
LIB "sample.lib";
↦     ? Library sample.lib: ERROR occurred: in line 26, 497.
↦     ? missing close bracket '}' at end of library in line 26.
↦     ? Cannot load library,... aborting.
↦     ? error occurred in STDIN line 1: `LIB "sample.lib";`
```

3.9.7 option(warn)

If this option is set some constructs which **may** lead to bug will result in a warning. While there are legitimate uses for them and they are **not errors** is is worth thinking about it.

change of options during a procedure call: is this side effect intended?

use of `def`: avoids type checking, but useful if a procedure handles several types at once

`ASSUME` outside of procedures: while a failed `ASSUME` aborts the current procedures and return to the top level - what should it do at top level?

See Section 5.1.111 [option], page 234.

3.10 Dynamic loading

In addition to the concept of libraries, it is also possible to dynamically extend the functionality by loading functions written in C/C++ or some other higher programming language. A collection of such functions is called a dynamic module and can be loaded by the command `LIB` or `load`. It is basically handled in the same way as a library: upon loading, a new `package` is created which holds the contents of the dynamic module. General information about the loaded module can be displayed by the command `help package_name`. After loading the dynamic module, its functions can be used exactly like the built-in SINGULAR functions.

To have the full functionality of a built-in function, dynamic modules need to comply with certain requirements on their internal structure. As this would be beyond the scope of the `Singular` manual, a separate, more detailed guide on how to write and use dynamic modules is available.

4 Data types

This chapter explains all data types of Singular in alphabetical order. For every type, there is a description of the declaration syntax as well as information about how to build expressions of certain types.

The term expression list in Singular refers to any comma separated list of expressions.

For the general syntax of a declaration see Section 3.5.1 [General command syntax], page 41.

4.1 cring

Variables of type cring represent the ring of coefficients (see Section 4.15 [number], page 115)

4.1.1 cring declarations

Syntax: `cring` name = cring_expression ;

Purpose: defines a new coefficient ring resp. field to be used for a ring definition (see Section 4.20 [ring], page 127). Most objects of this type are predefined.

Default: none

Example:

```
  ZZ;
↦ ZZ
  ZZ/3;
↦ ZZ/3
```

4.1.2 cring expressions

A cring expression is:

1. an identifier of type cring:

   QQ - the rational numbers

   ZZ - the integers

2. a function returning cring
3. an expression involving crings and the arithmetic operations `/`.

Example:

```
  ZZ/3;
↦ ZZ/3
```

See Section 4.20 [ring], page 127.

4.1.3 cring operations

`/` residue class ring

Example:

```
ZZ/101;
↦ ZZ/101
```

4.1.4 cring related functions

`crossprod`
: crooss product of several objects of type cring (see Section 5.1.15 [crossprod], page 168)

`Float`
: several variants of Floating point (inexact) real and complex numbers (see Section 5.1.45 [Float], page 186).

`flintQ`
: multivariate rational functions over Q (via flint, requires >=2.5.3) (see Section 5.1.44 [flintQ], page 186).

See Section 5.1.45 [Float], page 186; Section 5.1.15 [crossprod], page 168; Section 5.1.44 [flintQ], page 186.

4.2 bigint

Variables of type bigint represent the arbitrary long integers. They can only be constructed from other types (int, number).

4.2.1 bigint declarations

Syntax: `bigint` name = int_expression ;

Purpose: defines a long integer variable

Default: 0

Example:

```
  bigint i = 42;
  ring r=0,x,dp;
  number n=2;
  bigint j = i + bigint(n)^50; j;
↦ 1125899906842666
```

4.2.2 bigint expressions

A bigint expression is:

1. an identifier of type bigint
2. a function returning bigint
3. an expression involving bigints and the arithmetic operations `+`, `-`, `*`, `div`, `%` (`mod`), or `^`
4. a type cast to bigint.

Example:

```
// Note: 11*13*17*100*200*2000*503*1111*222222
// returns a machine integer:
11*13*17*100*200*2000*503*1111*222222;
↦ // ** int overflow(*), result may be wrong
↦ -6869239595516308480
// using the type cast number for a greater allowed range
bigint(11)*13*17*100*200*2000*503*1111*222222;
↦ 12075748128684240000000
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.7 [int], page 85; Section 4.15 [number], page 115.

4.2.3 bigint operations

`+` addition

`-` negation or subtraction

`*` multiplication

`div` integer division (omitting the remainder >= 0)

`mod`, `%` integer modulo (the remainder of the division `div`)

`^`, `**` exponentiation (exponent must be non-negative)

`<`, `>`, `<=`, `>=`, `==`, `<>`
comparators

Example:

```
  bigint(5)*2, bigint(2)^100-10;
  ↦ 10 1267650600228229401496703205366
  bigint(-5) div 2, bigint(-5) mod 2;
  ↦ -3 1
```

4.2.4 bigint related functions

`gcd` greatest common divisor (see Section 5.1.50 [gcd], page 190)

`memory` memory usage (see Section 5.1.89 [memory], page 219)

See Section 5.1.89 [memory], page 219.

4.3 bigintmat

Big integer matrices are matrices with big integer entries. No basering definition is required to use bigint matrices, for they do not belong to a ring. Bigintmat entries can have any size because of the use of bigint.

4.3.1 bigintmat declarations

Syntax: `bigintmat` name `=` bigintmat_expression `;`
`bigintmat` name `[` rows `]` `[` cols `]` `=` bigintmat_expression `;`
`bigintmat` name `[` rows `]` `[` cols `]` `=` list_of_int_and_bigint expressions `;`
rows and cols must be positive int expressions.

Purpose: defines a bigintmat variable.
Given a list of (big) integers, the matrix is filled up with the first row from the left to the right, then the second one and so on. If the (big-)int_list contains less than rows*cols elements, the remaining ones are set to zero; if it contains more elements, only the first rows*cols ones are considered.

Default: empty (1x0 matrix)

Example:

```
  bigintmat bim[4][3]=2, 5, 224553233465, 232444, 434, 0, 0, 4544232222;
  bim;
↦      2,         5,224553233465,
↦ 232444,       434,           0,
↦      0,4544232222,           0,
↦      0,         0,           0
  bim[2, 1];
↦ 232444
```

4.3.2 bigintmat expressions

A bigintmat expression is:

1. an identifier of type bigintmat
2. a function returning bigintmat
3. a bigintmat operation involving (big-)ints and int operations (+, -, *)
4. an expression involving bigintmats and the operations (+, -, *)
5. a type cast to bigintmat (see Section 4.3.3 [bigintmat type cast], page 76)

Example:

```
  bigintmat m1[2][2]=1, 2, 6, 3;
  m1*3;
↦  3,6,
↦ 18,9
  intmat im[3][2] = intmat(m1*3);
  bigintmat m2 = bigintmat(im); // cast intmat im to bigintmat
  m2;
↦  3,6,
↦ 18,9
  m2*m1+m2;
↦ 42,30,
↦ 90,72
  _+4;
↦ 46, 0,
↦  0,76
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.3 [bigintmat], page 75.

4.3.3 bigintmat type cast

Syntax: `bigintmat` (expression)

Type: bigintmat

Purpose: Converts expression to a bigintmat, where expression must be of type intmat, or bigintmat.The size (resp. dimension) of the created bigintmat equals the size (resp. dimension) of the expression.

Example:

```
  intmat im[2][1]=2, 3;
  bigintmat(im);
↦ 2,
↦ 3
  bigintmat(_);
```

```
↦ 2,
↦ 3
  bigintmat(intmat(intvec(1,2,3,4), 2, 2)); //casts at first to intmat, the
↦ 1,2,
↦ 3,4
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.3 [bigintmat], page 75; Section 4.8.3 [intmat type cast], page 92.

4.3.4 bigintmat operations

`+` addition with intmat, int, or bigint. In case of (big-)int, it is added to every entry of the matrix.

`-` negation or subtraction with intmat, int, or bigint. In case of (big-)int, it is subtracted from every entry of the matrix.

`*` multiplication with intmat, int, or bigint; In case of (big-)int, every entry of the matrix is multiplied by the (big-)int

`<>`, `==` comparators

bigintmat_expression `[ int, int ]`
: is a bigintmat entry, where the first index indicates the row and the second the column

Example:

```
  bigintmat m[3][4] = 3,3,6,3,5,2,2,7,0,0,45,3;
  m;
↦ 3,3, 6,3,
↦ 5,2, 2,7,
↦ 0,0,45,3
  m[1,3];                // show entry at [row 1, col 3]
↦ 6
  m[1,3] = 10;           // set entry at [row 1, col 3] to 10
  m;
↦ 3,3,10,3,
↦ 5,2, 2,7,
↦ 0,0,45,3
  size(m);          // number of entries
↦ 12
  bigintmat n[2][3] = 2,6,0,4,0,5;
  n * m;
↦ 36,18, 32,48,
↦ 12,12,265,27
  typeof(_);
↦ bigintmat
  -m;
↦ -3,-3,-10,-3,
↦ -5,-2, -2,-7,
↦  0, 0,-45,-3
  bigintmat o;
  o=n-10;
  o;
↦ -8,  0,0,
↦  0,-10,0
  m*2;           // double each entry of m
↦  6,6,20, 6,
↦ 10,4, 4,14,
↦  0,0,90, 6
  o-2*m;
↦    ? bigintmat/cmatrix not compatible
↦    ? error occurred in or before ./examples/bigintmat_operations.sing lin\
   e 15: ‘  o-2*m;‘
```

4.4 bigintvec

Big integer vectors are vectors with big integer entries. No basering definition is required to use bigint vectors, for they do not belong to a ring. Bigintvec entries can have any size because of the use of bigint.

4.4.1 bigintvec declarations

Syntax: `bigintvec` name = bigintvec_expression ;
`bigintvec` name] = list_of_int_and_bigint expressions ;

Purpose: defines a bigintvec variable.

Default: empty

Example:

```
  bigintvec v=2, 5, 224553233465, 232444, 434, 0, 0, 4544232222;
  v;
↦ 2,5,224553233465,232444,434,0,0,4544232222
  v[2];
↦ 5
```

4.4.2 bigintvec expressions

A bigintvec expression is:

1. an identifier of type bigintvec
2. a function returning bigintvec
3. a type cast to bigintvec (see Section 4.4.3 [bigintvec type cast], page 78)

Example:

```
  bigintmat m1[2][2]=1, 2, 6, 3;
  bigintvec v=m1[1,1..2];
  v;
↦ 1,2
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.4 [bigintvec], page 78.

4.4.3 bigintvec type cast

Syntax: `bigintvec ( `expression` )`

Type: bigintvec

Purpose: Converts expression to a bigintvec, where expression must be of type intvec, or bigintmat.The size of the created bigintvec equals the size of the expression.

Example:

```
          // TODO
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.4 [bigintvec], page 78.

4.4.4 bigintvec operations

`+` addition with intvec, int, or bigint. In case of (big-)int, it is added to every entry of the vector.

`-` negation or subtraction with intvec, int, or bigint. In case of (big-)int, it is subtracted from every entry of the matrix.

`*` multiplication with int, or bigint; every entry of the vector is multiplied by the (big-)int

`<>`, `==` comparators

bigintvec_expression `[` int `]`
is a bigintvec entry.

Example:

```
  bigintvec v = 3,3,6,3,5,2,2,7,0,0,45,3;
  v;
↦ 3,3,6,3,5,2,2,7,0,0,45,3
  v[3];               // show entry at [3]
↦ 6
  v[3] = 10;          // set entry at [3] to 10
  v;
↦ 3,3,10,3,5,2,2,7,0,0,45,3
  size(v);         // number of entries
↦ 12
  -v;
↦ -3,-3,-10,-3,-5,-2,-2,-7,0,0,-45,-3
  v-10;
↦ -7,0,0,0,0,0,0,0,0,0,0,0
  v*2;           // double each entry of v
↦ 6,6,20,6,10,4,4,14,0,0,90,6
```

4.5 def

Objects may be defined without a specific type: they inherit their type from the first assignment to them. E.g., `ideal i=x,y,z; def j=i^2;` defines the ideal `i^2` with the name `j`.

Note: Unlike other assignments a ring as an untyped object is not a copy but another reference to the same (possibly unnamed) ring. This means that entries in one of these rings appear also in the other ones. The following defines a ring `s` which is just another reference (or name) for the basering `r`. The name `basering` is an alias for the current ring.

```
  ring r=32003,(x,y,z),dp;
  poly f = x;
  def s=basering;
  setring s;
  nameof(basering);
↦ s
  listvar();
↦ // s                            [0]  *ring
↦ //      f                            [0]  poly
↦ // r                            [0]  ring(*)
  poly g = y;
  kill f;
  listvar(r);
↦ // r                            [0]  ring(*)
↦ // g                            [0]  poly
  ring t=32003,(u,w),dp;
  def rt=r+t;
  rt;
↦ // coefficients: ZZ/32003
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering dp
↦ //                  : names    u w
↦ //        block   3 : ordering C
```

This reference to a ring with def is useful if the basering is not local to the procedure (so it cannot be accessed by its name) but one needs a name for it (e.g., for a use with `setring` or `map`). `setring r;` does not work in this case, because `r` may not be local to the procedure.

4.5.1 def declarations

Syntax: `def` name `=` expression `;`

Purpose: defines an object of the same type as the right-hand side.

Default: none

Note: This is useful if the right-hand side may be of variable type as a consequence of a computation (e.g., ideal or module or matrix). It may also be used in procedures to give the basering a name which is local to the procedure.

Example:

```
  def i=2;
  typeof(i);
↦ int
```

See Section 5.1.161 [typeof], page 282.

4.6 ideal

Ideals are represented as lists of polynomials which generate the ideal. Like polynomials they can only be defined or accessed with respect to a basering.

Note: `size` counts only the non-zero generators of an ideal whereas `ncols` counts all generators; see Section 5.1.144 [size], page 264, Section 5.1.104 [ncols], page 231.

4.6.1 ideal declarations

Syntax: `ideal` name = list_of_poly_and_ideal_expressions ;
`ideal` name = ideal_expression ;

Purpose: defines an ideal.

Default: 0

Example:

```
  ring r=0,(x,y,z),dp;
  poly s1 = x2;
  poly s2 = y3;
  poly s3 = z;
  ideal i =  s1, s2-s1, 0,s2*s3, s3^4;
  i;
↦ i[1]=x2
↦ i[2]=y3-x2
↦ i[3]=0
↦ i[4]=y3z
↦ i[5]=z4
  size(i);
↦ 4
  ncols(i);
↦ 5
```

4.6.2 ideal expressions

An ideal expression is:

1. an identifier of type ideal
2. a function returning an ideal
3. a combination of ideal expressions by the arithmetic operations `+` or `*`
4. a power of an ideal expression (operator `^` or `**`)
   Note that the computation of the product `i*i` involves all products of generators of `i` while `i^2` involves only the different ones, and is therefore faster.
5. a type cast to ideal

Example:

```
  ring r=0,(x,y,z),dp;
  ideal m = maxideal(1);
  m;
↦ m[1]=x
↦ m[2]=y
↦ m[3]=z
  poly f = x2;
  poly g = y3;
  ideal i = x*y*z , f-g, g*(x-y) + f^4 ,0, 2x-z2y;
  ideal M = i + maxideal(10);
  timer =0;
  i = M*M;
  timer;
↦ 0
```

```
  ncols(i);
↦ 505
  timer =0;
  i = M^2;
  ncols(i);
↦ 505
  timer;
↦ 0
  i[ncols(i)];
↦ x20
  vector v = [x,y-z,x2,y-x,x2yz2-y];
  ideal j = ideal(v);
```

4.6.3 ideal operations

+ addition (concatenation of the generators and simplification)

* multiplication (with ideal, poly, vector, module; simplification in case of multiplication with ideal)

^ exponentiation (by a non-negative integer)

ideal_expression [intvec_expression]
are polynomial generators of the ideal, index 1 gives the first generator.

Note: For simplification of an ideal, see also Section 5.1.143 [simplify], page 263.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I = 0,x,0,1;
  I;
↦ I[1]=0
↦ I[2]=x
↦ I[3]=0
↦ I[4]=1
  I + 0;    // simplification
↦ _[1]=1
  ideal J = I,0,x,x-z;;
  J;
↦ J[1]=0
↦ J[2]=x
↦ J[3]=0
↦ J[4]=1
↦ J[5]=0
↦ J[6]=x
↦ J[7]=x-z
  I * J;   //  multiplication with simplification
↦ _[1]=1
  I*x;
↦ _[1]=0
↦ _[2]=x2
↦ _[3]=0
↦ _[4]=x
  vector V = [x,y,z];
  print(V*I);
↦ 0,x2,0,x,
↦ 0,xy,0,y,
↦ 0,xz,0,z
  ideal m = maxideal(1);
  m^2;
↦ _[1]=x2
↦ _[2]=xy
↦ _[3]=xz
↦ _[4]=y2
↦ _[5]=yz
↦ _[6]=z2
  ideal II = I[2..4];
  II;
↦ II[1]=x
↦ II[2]=0
↦ II[3]=1
```

4.6.4 ideal related functions

`char_series`
: irreducible characteristic series (see Section 5.1.6 [char_series], page 161)

`coeffs`
: matrix of coefficients (see Section 5.1.12 [coeffs], page 165)

`contract`
: contraction by an ideal (see Section 5.1.13 [contract], page 167)

`diff`
: partial derivative (see Section 5.1.24 [diff], page 173)

`degree`
: multiplicity, dimension and codimension of the ideal of leading terms (see Section 5.1.20 [degree], page 171)

`dim`
: Krull dimension of basering modulo the ideal of leading terms (see Section 5.1.25 [dim], page 174)

`eliminate`
: elimination of variables (see Section 5.1.28 [eliminate], page 176)

`facstd`
: factorizing Groebner basis algorithm (see Section 5.1.34 [facstd], page 179)

`factorize`
: ideal of factors of a polynomial (see Section 5.1.36 [factorize], page 180)

`fglm`
: Groebner basis computation from a Groebner basis w.r.t. a different ordering (see Section 5.1.39 [fglm], page 183)

`finduni`
: computation of univariate polynomials lying in a zero dimensional ideal (see Section 5.1.43 [finduni], page 185)

`fres`
: free resolution of a standard basis (see Section 5.1.48 [fres], page 188)

`groebner`
: Groebner basis computation (a wrapper around `std,stdhilb,stdfglm`,...) (see Section 5.1.53 [groebner], page 191)

`highcorner`
: the smallest monomial not contained in the ideal. The ideal has to be zero-dimensional. (see Section 5.1.55 [highcorner], page 194)

`homog`
: homogenization with respect to a variable (see Section 5.1.57 [homog], page 196)

`hilb` Hilbert series of a standard basis (see Section 5.1.56 [hilb], page 195)

`indepSet` sets of independent variables of an ideal (see Section 5.1.61 [indepSet], page 199)

`interred` interreduction of an ideal (see Section 5.1.64 [interred], page 201)

`intersect`
ideal intersection (see Section 5.1.65 [intersect], page 202)

`jacob` ideal of all partial derivatives resp. jacobian matrix (see Section 5.1.66 [jacob], page 203)

`jet` Taylor series up to a given order (see Section 5.1.68 [jet], page 204)

`kbase` vector space basis of basering modulo ideal of leading terms (see Section 5.1.69 [kbase], page 205)

`koszul` Koszul matrix (see Section 5.1.73 [koszul], page 207)

`lead` leading terms of a set of generators (see Section 5.1.75 [lead], page 209)

`lift` lift-matrix (see Section 5.1.80 [lift], page 211)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.81 [liftstd], page 212)

`lres` free resolution for homogeneous ideals (see Section 5.1.83 [lres], page 215)

`maxideal` power of the maximal ideal at 0 (see Section 5.1.88 [maxideal], page 219)

`minbase` minimal generating set of a homogeneous ideal, resp. module, or an ideal, resp. module, in a local ring (see Section 5.1.91 [minbase], page 221)

`minor` set of minors of a matrix (see Section 5.1.92 [minor], page 221)

`modulo` representation of $(h1+h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 5.1.94 [modulo], page 223)

`mres` minimal free resolution of an ideal resp. module w.r.t. a minimal set of generators of the given ideal resp. module (see Section 5.1.98 [mres], page 225)

`mstd` standard basis and minimal generating set of an ideal (see Section 5.1.100 [mstd], page 227)

`mult` multiplicity, resp. degree, of the ideal of leading terms (see Section 5.1.101 [mult], page 228)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`nres` a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 5.1.106 [nres], page 232)

`preimage` preimage under a ring map (see Section 5.1.117 [preimage], page 240)

`qhweight` quasihomogeneous weights of an ideal (see Section 5.1.124 [qhweight], page 246)

`quotient` ideal quotient (see Section 5.1.127 [quotient], page 248)

`reduce` normalform with respect to a standard basis (see Section 5.1.131 [reduce], page 251)

`res` free resolution of an ideal resp. module but not changing the given ideal resp. module (see Section 5.1.134 [res], page 253)

`simplify` simplification of a set of polynomials (see Section 5.1.143 [simplify], page 263)

`size` number of non-zero generators (see Section 5.1.144 [size], page 264)

`slimgb` Groebner basis computation with slim technique (see Section 5.1.145 [slimgb], page 265) `sortvec` permutation for sorting ideals resp. modules (see Section 5.1.146 [sortvec], page 266)

`sres` free resolution of a standard basis (see Section 5.1.149 [sres], page 269)

`std` standard basis computation (see Section 5.1.151 [std], page 271)

`stdfglm` standard basis computation with fglm technique (see Section 5.1.152 [stdfglm], page 272)

`stdhilb` Hilbert driven standard basis computation (see Section 5.1.153 [stdhilb], page 273)

`subst` substitution of a ring variable (see Section 5.1.154 [subst], page 274)

`syz` computation of the first syzygy module (see Section 5.1.156 [syz], page 280)

`vdim` vector space dimension of basering modulo ideal of leading terms (see Section 5.1.168 [vdim], page 286)

`weight` optimal weights (see Section 5.1.172 [weight], page 288)

4.7 int

Variables of type int represent the machine integers and are, therefore, limited in their range (e.g., the range is between -2147483647 and 2147483647 on 32-bit machines). They are mainly used to count things (dimension, rank, etc.), in loops (see Section 5.2.8 [for], page 295), and to represent boolean values (FALSE is represented by 0, every other value means TRUE, see Section 4.7.5 [boolean expressions], page 89).

Integers consist of a sequence of digits, possibly preceded by a sign. A space is considered as a separator, so it is not allowed between digits. A sequence of digits outside the allowed range is converted to the type `bigint`, see Section 4.2 [bigint], page 74.

4.7.1 int declarations

Syntax: `int` name = int_expression ;

Purpose: defines an integer variable.

Default: 0

Example:

```
  int i = 42;
  int j = i + 3; j;
↦ 45
  i = i * 3 - j; i;
↦ 81
  int k;   // assigning the default value 0 to k
  k;
↦ 0
```

4.7.2 int expressions

An int expression is:

1. a sequence of digits (if the number represented by this sequence is too large to fit into the range of integers it is automatically converted to the type number, if a basering is defined)
2. an identifier of type int
3. a function returning int

4. an expression involving ints and the arithmetic operations `+`, `-`, `*`, `div` (`/`), `%` (`mod`), or `^`
5. a boolean expression
6. a type cast to int

Note: Variables of type int represent the compiler integers and are, therefore, limited in their range (see Section 6.1 [Limitations], page 309). If this range is too small the expression must be converted to the type number over a ring with characteristic 0.

Example:

```
12345678901; // too large
↦ 12345678901
typeof(_);
↦ bigint
ring r=0,x,dp;
12345678901;
↦ 12345678901
typeof(_);
↦ bigint
// Note: 11*13*17*100*200*2000*503*1111*222222
// returns a machine integer:
11*13*17*100*200*2000*503*1111*222222;
↦ // ** int overflow(*), result may be wrong
↦ -6869239595516308480
// using the type cast number for a greater allowed range
number(11)*13*17*100*200*2000*503*1111*222222;
↦ 120757481286842400000000
ring rp=32003,x,dp;
12345678901;
↦ 12345678901
typeof(_);
↦ bigint
intmat m[2][2] = 1,2,3,4;
m;
↦ 1,2,
↦ 3,4
m[2,2];
↦ 4
typeof(_);
↦ int
det(m);
↦ -2
m[1,1] + m[2,1] == trace(m);
↦ 0
! 0;
↦ 1
1 and 2;
↦ 1
intvec v = 1,2,3;
def d =transpose(v)*v;    // scalarproduct gives an 1x1 intvec
typeof(d);
↦ intvec
int i = d[1];             // access the first (the only) entry in the intvec
```

```
ring rr=31,(x,y,z),dp;
poly f = 1;
i = int(f);               // cast to int
// Integers may be converted to constant  polynomials by an assignment,
poly g=37;
// define the constant polynomial g equal to the image of
// the integer 37 in the actual coefficient field, here it equals 6
g;
↦ 6
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.15 [number], page 115.

4.7.3 int operations

`++` changes its operand to its successor, is itself no int expression

`--` changes its operand to its predecessor, is itself no int expression

`+` addition

`-` negation or subtraction

`*` multiplication

`div` integer division (omitting the remainder), rounding toward 0

`%`, `mod` integer modulo (the remainder of the division

`^`, `**` exponentiation (exponent must be non-negative)

`<`, `>`, `<=`, `>=`, `==`, `<>`
comparators

Note: An assignment `j=i++;` or `j=i--;` is not allowed, in particular it does not change the value of `j`, see Section 6.1 [Limitations], page 309.

Example:

```
  int i=1;
  int j;
  i++; i;  i--; i;
↦ 2
↦ 1
  // ++ and -- do not return a value as in C, cannot assign
  j = i++;
↦ // ** right side is not a datum, assignment ignored
↦ // ** in line >>  j = i++;<<
  // the value of j is unchanged
  j; i;
↦ 0
↦ 2
  i+2, 2-i, 5^2;
↦ 4 0 25
  5 div 2, 8%3;
↦ 2 2
  -5 div 2, -5 mod 2, -5 % 2;
↦ -2 -1 -1
  1<2, 2<=2;
↦ 1 1
```

4.7.4 int related functions

`char` characteristic of the coefficient field of a ring (see Section 5.1.5 [char], page 161)

`deg` degree of a polynomial resp. vector (see Section 5.1.19 [deg], page 170)

`det` determinant (see Section 5.1.23 [det], page 172)

`dim` Krull dimension of basering modulo ideal of leading terms, resp. dimension of module of leading terms (see Section 5.1.25 [dim], page 174)

`extgcd` Bezout representation of gcd (see Section 5.1.33 [extgcd], page 178)

`find` position of a substring in a string (see Section 5.1.42 [find], page 185)

`gcd` greatest common divisor (see Section 5.1.50 [gcd], page 190)

`koszul` Koszul matrix (see Section 5.1.73 [koszul], page 207)

`memory` memory usage (see Section 5.1.89 [memory], page 219)

`mult` multiplicity of an ideal, resp. module, of leading terms (see Section 5.1.101 [mult], page 228)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`npars` number of ring parameters (see Section 5.1.105 [npars], page 231)

`nrows` number of rows of a matrix, resp. the rank of the free module where the vector or module lives (see Section 5.1.107 [nrows], page 232)

`nvars` number of ring variables (see Section 5.1.109 [nvars], page 233)

`ord` degree of the leading term of a polynomial resp. vector (see Section 5.1.112 [ord], page 238)

`par` n-th parameter of the basering (see Section 5.1.114 [par], page 239)

`pardeg` degree of a number considered as a polynomial in the ring parameters (see Section 5.1.115 [pardeg], page 239)

`prime` the next lower prime (see Section 5.1.118 [prime], page 241)

`random` a pseudo random integer between the given limits (see Section 5.1.128 [random], page 249)

`regularity`
regularity of a resolution (see Section 5.1.132 [regularity], page 252)

`rvar` test, if the given expression or string is a ring variable (see Section 5.1.139 [rvar], page 258)

`size` number of elements in an object (see Section 5.1.144 [size], page 264)

`trace` trace of an integer matrix (see Section 5.1.158 [trace], page 281)

`var` n-th ring variable of the basering (see Section 5.1.165 [var], page 285)

`vdim` vector space dimension of basering modulo ideal of leading terms, resp. of freemodule modulo module of leading terms (see Section 5.1.168 [vdim], page 286)

4.7.5 boolean expressions

A boolean expression is an int expression used in a logical context:

An int expression <> 0 evaluates to *TRUE* (represented by 1), 0 evaluates to *FALSE* (represented by 0).

The following is the list of available comparisons of objects of the same type.

Note: There are no comparisons for ideals and modules, resolutions and maps.

1. integer comparisons:

```
i == j
i != j    // or     i <> j
i <= j
i >= j
i > j
i < j
```

2. number comparisons:

```
m == n
m != n    // or     m <> n
m < n
m > n
m <= n
m >= n
```

   For numbers from Z/p or from field extensions not all operations are useful:
   - 0 is always the smallest element,
   - in Z/p the representatives in the range -(p-1)/2..(p-1)/2 when p>2 resp. 0 and 1 for p=2 are used for comparisons,
   - in field extensions the last two operations (`>=`,`<=`) yield always TRUE (1) and the `<` and `>` are equivalent to `!=`.

3. polynomial or vector comparisons:

```
f == g
f != g    // or     f <> g
f <= g    // comparing the leading term w.r.t. the monomial order
f <  g
f >= g
f >  g
```

4. intmat or matrix comparisons:

```
v == w
v != w    // or     v <> w
```

5. intvec or string comparisons:

```
f == g
f != g    // or     f <> g
f <= g    // comparing lexicographically
f >= g    // w.r.t. the order specified by ASCII
f >  g
f <  g
```

6. boolean expressions combined by boolean operations (`and`, `or`, `not`)

Note: All arguments of a logical expression are first evaluated and then the value of the logical expression is determined. For example, the logical expression `(a || b)` is evaluated by first evaluating `a` *and* `b`, even though the value of `b` has no influence on the value of `(a || b)`, if `a` evaluates to true.

Note that this evaluation is different from the left-to-right, conditional evaluation of logical expressions (as found in most programming languages). For example, in these other languages, the value of `(1 || b)` is determined without ever evaluating `b`.

See Section 6.3 [Major differences to the C programming language], page 310.

4.7.6 boolean operations

`and` logical `and`, may also be written as `&&`

`or` logical `or`, may also be written as `||`

`not` logical `not`, may also be written as `!`

The precedence of the boolean operations is:

1. parentheses
2. comparisons
3. not
4. and
5. or

Example:

```
    (1>2) and 3;
  ↦ 0
    1 > 2 and 3;
  ↦ 0
    ! 0 or 1;
  ↦ 1
    !(0 or 1);
  ↦ 0
```

4.8 intmat

Integer matrices are matrices with integer entries. For the range of integers see Section 6.1 [Limitations], page 309. Integer matrices do not belong to a ring, they may be defined without a basering being defined. An intmat can be multiplied by and added to an int; in this case the int is converted into an intmat of the right size with the integer on the diagonal. The integer `1`, for example, is converted into the unit matrix.

4.8.1 intmat declarations

Syntax: `intmat` name `=` intmat_expression `;`
`intmat` name `[` rows `] [` cols `] =` intmat_expression `;`
`intmat` name `[` rows `] [` cols `] =` list_of_int_and_intvec_and_intmat_expressions `;`
rows and cols must be positive int expressions.

Purpose: defines an intmat variable.
Given a list of integers, the matrix is filled up with the first row from the left to the right, then the second row and so on. If the int_list contains less than rows*cols elements, the matrix is filled up with zeros; if it contains more elements, only the first rows*cols elements are used.

Default: 0 (1 x 1 matrix)

Example:

```
  intmat im[3][5]=1,3,5,7,8,9,10,11,12,13;
  im;
↦ 1,3,5,7,8,
↦ 9,10,11,12,13,
↦ 0,0,0,0,0
  im[3,2];
↦ 0
  intmat m[2][3] = im[1..2,3..5];  // defines a submatrix
  m;
↦ 5,7,8,
↦ 11,12,13
```

4.8.2 intmat expressions

An intmat expression is:

1. an identifier of type intmat
2. a function returning intmat
3. an intmat operation involving ints and int operations (+, -, *, div, %)
4. an expression involving intmats and the operations (+, -, *)
5. a type cast to intmat (see Section 4.8.3 [intmat type cast], page 92)

Example:

```
  intmat Idm[2][2];
  Idm +1;          // add the unit intmat
↦ 1,0,
↦ 0,1
  intmat m1[3][2] = _,1,-2;  // take entries from the last result
  m1;
↦ 1,0,
↦ 0,1,
↦ 1,-2
  intmat m2[2][3]=1,0,2,4,5,1;
  transpose(m2);
↦ 1,4,
↦ 0,5,
↦ 2,1
  intvec v1=1,2,4;
  intvec v2=5,7,8;
  m1=v1,v2;         // fill m1 with v1 and v2
  m1;
↦ 1,2,
↦ 4,5,
↦ 7,8
  trace(m1*m2);
↦ 56
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.15 [number], page 115.

4.8.3 intmat type cast

Syntax: `intmat` (expression)
`intmat` (expression, int_n, int_m)

Type: intmat

Purpose: Converts expression to an intmat, where expression must be of type intvec, intmat, or bigintmat. If int_n and int_m are supplied, then they specify the dimension of the intmat. Otherwise, the size (resp. dimensions) of the intmat are determined by the size (resp. dimensions) of the expression. If expression is a bigintmat containing an entry larger the the limit of int, it is set to 0 in the returning intmat.

Example:

```
  intmat(intvec(1));
↦ 1
  intmat(intvec(1), 1, 2);
↦ 1,0
  intmat(intvec(1,2,3,4), 2, 2);
↦ 1,2,
↦ 3,4
  intmat(_, 2, 3);
↦ 1,2,3,
↦ 4,0,0
  intmat(_, 2, 1);
↦ 1,2
  bigintmat bim[2][3]=34, 64, 345553234, 35553, 6434, 6563335675;
  intmat(bim);
↦ 34,64,345553234,
↦ 35553,6434,-2026598917
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.8 [intmat], page 90; Section 4.13.3 [matrix type cast], page 110.

4.8.4 intmat operations

`+` addition with intmat or int; the int is converted into a diagonal intmat

`-` negation or subtraction with intmat or int; the int is converted into a diagonal intmat

`*` multiplication with intmat, intvec, or int; the int is converted into a diagonal intmat

`div,/` division of entries in the integers (omitting the remainder)

`%, mod` entries modulo int (remainder of the division)

`<>, ==` comparators

intmat_expression `[` intvec_expression`,` intvec_expression `]`
is an intmat entry, where the first index indicates the row and the second the column

Example:

```
  intmat m[2][4] = 1,0,2,4,0,1,-1,0,3,2,1,-2;
  m;
↦ 1,0,2,4,
↦ 0,1,-1,0
```

```
  m[2,3];          // entry at row 2, col 3
↦ -1
  size(m);         // number of entries
↦ 8
  intvec v = 1,0,-1,2;
  m * v;
↦ 7,1
  typeof(_);
↦ intvec
  intmat m1[4][3] = 0,1,2,3,v,1;
  intmat m2 = m * m1;
  m2;             //  2 x 3 intmat
↦ -2,5,4,
↦ 4,-1,-1
  m2*10;           // multiply each entry of m with 10;
↦ -20,50,40,
↦ 40,-10,-10
  -m2;
↦ 2,-5,-4,
↦ -4,1,1
  m2 % 2;
↦ 0,1,0,
↦ 0,1,1
  m2 div 2;
↦ -1,2,2,
↦ 2,-1,-1
  m2[2,1];          // entry at row 2, col 1
↦ 4
  m1[2..3,2..3];   // submatrix
↦ 1 0 2 1
  m2[nrows(m2),ncols(m2)];      // the last entry of intmat m2
↦ -1
```

4.8.5 intmat related functions

`betti` Betti numbers of a free resolution (see Section 5.1.4 [betti], page 159)

`det` determinant (see Section 5.1.23 [det], page 172)

`ncols` number of cols (see Section 5.1.104 [ncols], page 231)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232)

`random` pseudo random intmat (see Section 5.1.128 [random], page 249)

`size` total number of entries (see Section 5.1.144 [size], page 264)

`transpose`
transpose of an intmat (see Section 5.1.159 [transpose], page 282)

`trace` trace of an intmat (see Section 5.1.158 [trace], page 281)

4.9 intvec

Variables of type intvec are lists of integers. For the range of integers see Section 6.1 [Limitations], page 309. They may be used for simulating sets of integers (and other sets if the intvec is used as an index set for other objects). Addition and subtraction of an intvec with an int or an intvec is done element-wise.

4.9.1 intvec declarations

Syntax: `intvec` name = intvec_expression ;
`intvec` name = list_of_int_and_intvec_expressions ;

Purpose: defines an intvec variable.
An intvec consists of an ordered list of integers.

Default: 0

Example:

```
  intvec iv=1,3,5,7,8;
  iv;
↦ 1,3,5,7,8
  iv[4];
↦ 7
  iv[3..size (iv)];
↦ 5 7 8
```

4.9.2 intvec expressions

An intvec expression is:

1. a range: int expression `..` int expression
2. a repeated entry: int expression `:` positive int expression
   (`a:b` generates an `intvec` of length `b`>0 with identical entries `a`)
3. a function returning intvec
4. an expression involving intvec operations with int (`+`, `-`, `*`, `/`, `%`)
5. an expression of intvecs involving intvec operations (`+`, `-`)
6. an expression involving an intvec operation with intmat (`*`)
7. a type cast to intvec

Example:

```
  intvec v=-1,2;
  intvec w=v,v;         // concatenation
  w;
↦ -1,2,-1,2
  w=2:3;                // repetition
  w;
↦ 2,2,2
  int k = 3;
  v = 7:k;
  v;
↦ 7,7,7
  v=-1,2;
  w=-2..2,v,1;
  w;
↦ -2,-1,0,1,2,-1,2,1
  intmat m[3][2] = 0,1,2,-2,3,1;
  m*v;
↦ 2,-6,-1
  typeof(_);
↦ intvec
  v = intvec(m);
  v;
↦ 0,1,2,-2,3,1
  ring r;
  poly f = x2z + 2xy-z;
  f;
↦ x2z+2xy-z
  v = leadexp(f);
  v;
↦ 2,0,1
```

4.9.3 intvec operations

`+` addition with intvec or int (component-wise)

`-` negation or subtraction with intvec or int (component-wise)

`*` multiplication with int (component-wise)

`/`, `div` division by int (component-wise)

`%`, `mod` modulo (component-wise)

<>, ==, <=, >=, >, <
comparison (done lexicographically, different length will be filled with 0 at th right)

intvec_expression `[` int_expression `]`
is an element of the intvec; the first element has index one.

Example:

```
  intvec iv =  1,3,5,7,8;
  iv+1;               // add 1 to each entry
↦ 2,4,6,8,9
  iv*2;
↦ 2,6,10,14,16
  iv;
↦ 1,3,5,7,8
  iv-10;
↦ -9,-7,-5,-3,-2
  iv=iv,0;
  iv;
↦ 1,3,5,7,8,0
  iv div 2;
↦ 0,1,2,3,4,0
  iv+iv;              // component-wise addition
↦ 2,6,10,14,16,0
  iv[size(iv)-1];     // last-1 entry
↦ 8
  intvec iw=2,3,4,0;
  iv==iw;             // lexicographic comparison
↦ 0
```

```
  iv < iw;
↦ 1
  iv != iw;
↦ 1
  iv[2];
↦ 3
  iw = 4,1,2;
  iv[iw];
↦ 7 1 3
```

4.9.4 intvec related functions

`hilb` Hilbert series as intvec (see Section 5.1.56 [hilb], page 195)

`indepSet` sets of independent variables of an ideal (see Section 5.1.61 [indepSet], page 199)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.77 [leadexp], page 210)

`monomial` the power product corresponding to the exponent vector (see Section 5.1.96 [monomial], page 225)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232)

`qhweight` quasihomogeneous weights (see Section 5.1.124 [qhweight], page 246)

`size` length of the intvec (see Section 5.1.144 [size], page 264)

`sortvec` permutation for sorting ideals/modules (see Section 5.1.146 [sortvec], page 266)

`transpose`
transpose of an intvec, returns an intmat (see Section 5.1.159 [transpose], page 282)

`weight` weights for the weighted ecart method (see Section 5.1.172 [weight], page 288)

4.10 link

Links are the communication channels of SINGULAR, i.e., something SINGULAR can write to and/or read from. Currently, SINGULAR supports four different link types:

- ASCII links (see Section 4.10.4 [ASCII links], page 98)
- ssi links (see Section 4.10.5 [Ssi links], page 99)
- pipe links (see Section 4.10.6 [Pipe links], page 101)
- DBM links (see Section 4.10.7 [DBM links], page 102)

4.10.1 link declarations

Syntax: `link` name = string_expression ;

Purpose: defines a new communication link.

Default: none

Example:

```
link l=":w example.txt";
int i=22;          // cf. ASCII links for explanation
string s="An int follows:";
write(l,s,i);
l;
```

```
↦ // type : ASCII
↦ // mode : w
↦ // name : example.txt
↦ // open : yes
↦ // read : not ready
↦ // write: ready
  close(l);          //
  read(l);
↦ An int follows:
↦ 22
↦
  close(l);
```

4.10.2 link expressions

A link expression is:

1. an identifier of type link
2. a string describing the link

A link is described by a string which consists of two parts: a property string followed by a name string. The property string describes the type of the link (`ASCII`, `ssi` or `DBM`) and the mode of the link (e.g., open for read, write or append). The name string describes the filename of the link, resp. a network connection for ssi links.

For a detailed format description of the link describing string see:

- for ASCII links: Section 4.10.4 [ASCII links], page 98
- ssi links (see Section 4.10.5 [Ssi links], page 99)
- pipe links (see Section 4.10.6 [Pipe links], page 101)
- for DBM links: Section 4.10.7 [DBM links], page 102

4.10.3 link related functions

`close` closes a link (see Section 5.1.10 [close], page 163)

`dump` generates a dump of all variables and their values (see Section 5.1.27 [dump], page 175)

`getdump` reads a dump (see Section 5.1.52 [getdump], page 191)

`open` opens a link (see Section 5.1.110 [open], page 234)

`read` reads from a link (see Section 5.1.130 [read], page 250)

`status` gets the status of a link (see Section 5.1.150 [status], page 270)

`write` writes to a link (see Section 5.1.174 [write], page 289)

`kill` closes and kills a link (see Section 5.1.71 [kill], page 206)

`waitall` wait till all links of a list of links become ready (only ssi:tcp links) (see Section 5.1.169 [waitall], page 286)

`waitfirst`

wait till at least one link of a list of links become ready (only ssi:tcp links) (see Section 5.1.170 [waitfirst], page 287)

4.10.4 ASCII links

Via ASCII links data that can be converted to a string can be written into files for storage or communication with other programs. The data is written in plain ASCII format. The output format of polynomials is done w.r.t. the value of the global variable `short` (see Section 5.3.7 [short], page 305). Reading from an ASCII link returns a string — conversion into other data is up to the user. This can be done, for example, using the command `execute` (see Section 5.1.32 [execute], page 178).

The ASCII link describing string has to be one of the following:

1. `"ASCII: "` + filename
   the mode (read or append) is set by the first `read` or `write` command.
2. `"ASCII:r "` + filename
   opens the file for reading.
3. `"ASCII:w "` + filename
   opens the file for overwriting.
4. `"ASCII:a "` + filename
   opens the file for appending.

There are the following default values:

- the type `ASCII` may be omitted since ASCII links are the default links.
- if non of `r`, `w`, or `a` is specified, the mode of the link is set by the first `read` or `write` command on the link. If the first command is `write`, the mode is set to `a` (append mode).
- if the filename is omitted, `read` reads from stdin and `write` writes to stdout.

Using these default rules, the string `":r temp"` describes a link which is equivalent to the link `"ASCII:r temp"`: an ASCII link to the file `temp` which is opened for reading. The string `"temp"` describes an ASCII link to the file `temp`, where the mode is set by the first `read` or `write` command. See also the example below.

Note that the filename may contain a path. On Microsoft Windows (resp. MS-DOS) platforms, names of a drive can precede the filename, but must be started with a `//` (as in `//c/temp/ex`. An ASCII link can be used either for reading or for writing, but not for both at the same time. A `close` command must be used before a change of I/O direction. Types without a conversion to `string` cannot be written.

Example:

```
    ring r=32003,(x,y,z),dp;
    link l=":w example.txt";     // type is ASCII, mode is overwrite
    l;
  ↦ // type : ASCII
  ↦ // mode : w
  ↦ // name : example.txt
  ↦ // open : no
  ↦ // read : not ready
  ↦ // write: not ready
    status(l, "open", "yes");    // link is not yet opened
  ↦ 0
    ideal i=x2,y2,z2;
    write (l,1,";",2,";","ideal i=",i,";");
    status(l, "open", "yes");    // now link is open
  ↦ 1
    status(l, "mode");           // for writing
  ↦ w
    close(l);                    // link is closed
    write("example.txt","int j=5;");// data is appended to file
    read("example.txt");         // data is returned as string
  ↦ 1
  ↦ ;
  ↦ 2
  ↦ ;
  ↦ ideal i=
  ↦ x2,y2,z2;
  ↦ int j=5;
  ↦
    execute(read(l));            // read string is executed
  ↦ 1
  ↦ 2
  ↦ // ** redefining i (ideal i=) ./examples/ASCII_links.sing:14
    close(l);                    // link is closed
```

4.10.5 Ssi links

Ssi (simple singular interface) links give the possibility to store and communicate data betweenm Singular processes: Read and write access is very fast compared to ASCII links. Ssi links can be established using files or using TCP sockets. For ring-dependent data, a ring description is written together with the data. Reading from an Ssi link returns an expression (not a string) which was evaluated after the read operation. If the expression read from an Ssi link is not from the same ring as the current ring, then a `read` changes the current ring.

Currently under development - not everything is implemtented.

4.10.5.1 Ssi file links

Ssi file links provide the possibility to store data in a file using the ssi format. For storing large amounts of data, ssi file links should be used instead of ASCII links. Unlike ASCII links, data read from ssi file links is returned as expressions one at a time.

The ssi file link describing string has to be one of the following:

1. `"ssi:r "` + filename
   opens the file for reading.
2. `"ssi:w "` + filename
   opens the file for overwriting.
3. `"ssi:a "` + filename
   opens the file for appending.

Note that the filename may contain a path. An ssi file link can be used either for reading or for writing, but not for both at the same time. A `close` command must be used before a change of I/O direction.

Example:

```
  ring r;
  link l="ssi:w example.ssi"; // type=ssi, mode=overwrite
  l;
↦ // type : ssi
↦ // mode : w
↦ // name : example.ssi
↦ // open : no
↦ // read : not open
↦ // write: not open
  ideal i=x2,y2,z2;
  write (l,1, i, "hello world");// write three expressions
  write(l,4);                   // append one more expression
  close(l);                     // link is closed
  // open the file for reading now
  read(l);                      // only first expression is read
↦ 1
  kill r;                       // no basering active now
  def i = read(l);              // second expression
  // notice that current ring was set, the name was assigned
  // automatically
  listvar(ring);
↦ // ssiRing0                        [0]  *ring
↦ // ZZ                              [0]  cring
↦ // QQ                              [0]  cring
  def s = read(l);              // third expression
  listvar();
↦ // s                               [0]  string hello world
↦ // ssiRing0                        [0]  *ring
↦ //      i                               [0]  ideal, 3 generator(s)
↦ // l                               [0]  link
  close(l);                     // link is closed
```

4.10.5.2 Ssi tcp links

Ssi tcp links give the possibility to exchange data between two processes which may run on the same or on different computers. Ssi tcp links can be opened in four different modes:

`tcp` Singular acts as a server.

`connect` Singular acts as a client.

`tcp <host>:<program>`
Singular acts as a client, launching an application as server. This reuires `ssh/sshd` to be installed an the computers (and preferably an automatic login via `.ssh/authorized_keys`).

`fork` Singular acts as a client, forking another Singular as server.

The Ssi tcp link describing string has to be

- tcp mode:
  1. `"ssi:tcp"`

  Singular becomes a server and waits at the first free port (>1024) for a connect call.
- connect mode:
  2. `"ssi:connect "` + host:port

  Singular becomes a client and connects to a server waiting at the host and port.
- launch mode:
  4. `"ssi:tcp"` + host:application

Singular becomes a client and starts (launches) the application using ssh on a (possibly) different host which then acts as a server.

- fork mode:

  8. `"ssi:fork"`

  Singular becomes a client and forks another Singular on the same host which acts as a server.

To open an ssi tcp link in launch mode, the application to launch must either be given with an absolute pathname, or must be in a directory contained in the search path. The launched application acts as a server, whereas the Singular that actually opened the link acts as a client. The client "listens" at the some free port until the server application does a connect call.

If the ssi tcp link is opened in fork mode a child of the current Singular is forked. All variables and their values are inherited by the child. The child acts as a server whereas the Singular that actually opened the link acts as a client.

To arrange the evaluation of an expression by a server, the expression must be quoted using the command `quote` (see Section 5.1.126 [quote], page 247), so that a local evaluation is prevented. Otherwise, the expression is evaluated first, and the result of the evaluation is written, instead of the expression which is to be evaluated.

If Singular is in server mode, the value of the variable `link_ll` is the ssi link connecting to the client and Singular is in an infinite read-eval-write loop until the connection is closed from the client side (by closing its connecting link). Reading and writing is done to the link `link_ll`: After an expression is read, it is evaluated and the result of the evaluation is written back. That is, for each expression which was written to the server, there is exactly one expression written back. This might be an "empty" expression, if the evaluation on the server side does not return a value.

Ssi tcp links should explicitly be opened before being used. Ssi tcp links are bidirectional, i.e. can be used for both, writing and reading. Reading from an ssi tcp link blocks until data was written to that link. The `status` command can be used to check whether there is data to read.

Example:

```
    int i=7;
    link l = "ssi:fork";      // fork link declaration
    open(l); l;
  ↦ // type : ssi
  ↦ // mode : fork
  ↦ // name :
  ↦ // open : yes
  ↦ // read : not ready
  ↦ // write: ready

    write(l,quote(i)); // Child inherited vars and their values
    read(l);
  ↦ 7
    close(l);          // shut down forked child
```

4.10.6 Pipe links

Pipe links provide access to stdin and stdout of any program. Pipe links are bidirectional. **Syntax:** `"|: "` + string_for_system

The string_for system will be passed to `system` after conneting the input and output to the corresponding stdout and stdin.

Example:

```
       link l="|: date";
       open(l); l;
     ↦ // type : pipe
     ↦ // mode :
     ↦ // name : date
     ↦ // open : yes
     ↦ // read : not ready
     ↦ // write: ready
       read(l);
     ↦ Di 28. Nov 10:03:12  2023
       l;
     ↦ // type : pipe
     ↦ // mode :
     ↦ // name : date
     ↦ // open : yes
     ↦ // read : not ready
     ↦ // write: ready
       close(l);
```

4.10.7 DBM links

DBM links provide access to data stored in a data base. Each entry in the data base consists of a (key_string, value_string) pair. Such a pair can be inserted with the command `write(`link`,` key_string`,` value_string`)`. By calling `write(`link`,` key_string`)`, the entry with key key_string is deleted from the data base. The value of an entry is returned by the command `read(`link`,` key_string`)`. With only one argument, `read(`link`)` returns the next key in the data base. Using this feature a data base can be scanned in order to access all entries of the data base.

If a data base with name `name` is opened for writing for the first time, two files (`name.pag` and `name.dir`), which contain the data base, are automatically created.

The DBM link describing string has to be one of the following:

1. `"DBM: " +` name
   opens the data base for reading (default mode).
2. `"DBM:r " +` name
   opens the data base for reading.
3. `"DBM:rw " +` name
   opens the data base for reading and writing.

Note that `name` must be given without the suffix `.pag` or `.dir`. The name may contain an (absolute) path.

Example:

```
       link l="DBM:rw example";
       write(l,"1","abc");
       write(l,"3","XYZ");
       write(l,"2","ABC");
       l;
     ↦ // type : DBM
     ↦ // mode : rw
     ↦ // name : example
     ↦ // open : yes
```

```
↦ // read : ready
↦ // write: ready
  close(l);
  // read all keys (till empty string):
  read(l);
↦ 1
  read(l);
↦ 3
  read(l);
↦ 2
  read(l);
↦ 
  // read data corresponding to key "1"
  read(l,"1");
↦ abc
  // read all data:
  read(l,read(l));
↦ abc
  read(l,read(l));
↦ XYZ
  read(l,read(l));
↦ ABC
  // close
  close(l);
```

4.11 list

Lists are arrays whose elements can be of different types (including ring). If one element belongs to a ring the whole list belongs to that ring. This applies also to the special list `#`. The expression `list()` is the empty list.

Note that a list stores the objects itself and not the names. Hence, if `L` is a list, `L[1]` for example has no name. A name, say `R`, can be created for `L[1]` by `def R=L[1];`. To store also the name of an object, say `r`, it can be added to the list with `nameof(r);`. Rings may be objects of a list.

Note: Unlike other assignments a ring as an element of a list is not a copy but another reference to the same ring.

4.11.1 list declarations

Syntax: `list` name = expression_list;
`list` name = list_expression;

Purpose: defines a list (of objects of possibly different types).

Default: empty list

Example:

```
  list l=1,"str";
  l[1];
↦ 1
  l[2];
↦ str
  ring r;
  listvar(r);
```

```
↦ // r                              [0]  *ring
  ideal i = x^2, y^2 + z^3;
  l[3] = i;
  l;
↦ [1]:
↦    1
↦ [2]:
↦    str
↦ [3]:
↦    _[1]=x2
↦    _[2]=z3+y2
  listvar(r);     // the list l belongs now to the ring r
↦ // r                              [0]  *ring
↦ // l                              [0]  list, size: 3
↦ // i                              [0]  ideal, 2 generator(s)
```

4.11.2 list expressions

A list expression is:

1. the empty list `list()`
2. an identifier of type list
3. a function returning list
4. list expressions combined by the arithmetic operation `+`
5. a type cast to list

See Section 3.5.5 [Type conversion and casting], page 46.

Example:

```
  list l = "hello",1;
  l;
↦ [1]:
↦    hello
↦ [2]:
↦    1
  l = list();
  l;
↦ empty list
  ring r =0,x,dp;
  factorize((x+1)^2);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+1
↦ [2]:
↦    1,2
  list(1,2,3);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    3
```

4.11.3 list operations

`+` concatenation

`delete` deletes one element from list, returns new list

`insert` inserts or appends a new element to list, returns a new list

list_expression `[` int_expression `]`
is a list entry; the index 1 gives the first element.

Example:

```
  list l1 = 1,"hello",list(-1,1);
  list l2 = list(1,5,7);
  l1 + l2;             // a new list
↦ [1]:
↦    1
↦ [2]:
↦    hello
↦ [3]:
↦    [1]:
↦       -1
↦    [2]:
↦       1
↦ [4]:
↦    1
↦ [5]:
↦    5
↦ [6]:
↦    7
  l2 = delete(l2, 2);  // delete 2nd entry
  l2;
↦ [1]:
↦    1
↦ [2]:
↦    7
```

4.11.4 list related functions

`bareiss` returns a list of a matrix (lower triangular) and of an intvec (permutations of columns, see Section 5.1.3 [bareiss], page 158)

`betti` Betti numbers of a resolution (see Section 5.1.4 [betti], page 159)

`delete` deletion of an element from a list (see Section 5.1.21 [delete], page 171)

`facstd` factorizing Groebner basis algorithm (see Section 5.1.34 [facstd], page 179)

`factorize`
list of factors of a polynomial (see Section 5.1.36 [factorize], page 180)

`insert` insertion of a new element into a list (see Section 5.1.62 [insert], page 200)

`minres` minimization of a free resolution (see Section 5.1.93 [minres], page 223)

`names` list of all user-defined variable names (see Section 5.1.103 [names], page 229) `size` number of entries (see Section 5.1.144 [size], page 264)

`conversion from resolution`
(see Section 4.19 [resolution], page 125)

4.12 map

Maps are ring maps from a preimage ring into the basering.

Note:

- The target of a map is **ALWAYS** the actual basering
- The preimage ring has to be stored "by its name", that means, maps can only be used in such contexts, where the name of the preimage ring can be resolved (this has to be considered in subprocedures). See also Section 6.5 [Identifier resolution], page 315, Section 3.7.4 [Names in procedures], page 54.

Maps between rings with different coefficient fields are possible and listed below.

Canonically realized are

- $Q \to Q(a,\ldots)$ (Q : the rational numbers)
- $Q \to R$ (R : the real numbers)
- $Q \to C$ (C : the complex numbers)
- $Z/p \to (Z/p)(a,\ldots)$ (Z : the integers)
- $Z/p \to GF(p^n)$ (GF : the Galois field)
- $Z/p \to R$
- $R \to C$

Possible are furthermore

- $Z/p \to Q, \quad [i]_p \mapsto i \in [-p/2,\, p/2] \subseteq Z$
- $Z/p \to Z/p', \quad [i]_p \mapsto i \in [-p/2,\, p/2] \subseteq Z,\ i \mapsto [i]_{p'} \in Z/p'$
- $C \to R,$ by taking the real part

Finally, in Singular we allow the mapping from rings with coefficient field Q to rings whose ground fields have finite characteristic:

- $Q \to Z/p$
- $Q \to (Z/p)(a,\ldots)$

In these cases the denominator and the numerator of a number are mapped separately by the usual map from Z to Z/p, and the image of the number is built again afterwards by division. It is thus not allowed to map numbers whose denominator is divisible by the characteristic of the target ground field, or objects containing such numbers. We, therefore, strongly recommend using such maps only to map objects with integer coefficients.

4.12.1 map declarations

Syntax: `map` name = preimage_ring_name , ideal_expression ;
`map` name = preimage_ring_name , list_of_poly_and_ideal_expressions ;
`map` name = map_expression ;

Purpose: defines a ring map from preimage_ring to basering.
Maps the variables of the preimage ring to the generators of the ideal. If the ideal contains less elements than variables in the preimage_ring the remaining variables are

mapped to 0, if the ideal contains more elements these are ignored. The image ring is always the current basering. For the mapping of coefficients from different fields see Section 4.12 [map], page 106.

Default: none

Note: There are standard mappings for maps which are close to the identity map: `fetch` and `imap`.

The name of a map serves as the function which maps objects from the preimage_ring into the basering. These objects must be defined by names (no evaluation in the preimage ring is possible).

Example:

```
  ring r1=32003,(x,y,z),dp;
  ideal i=x,y,z;
  ring r2=32003,(a,b),dp;
  map f=r1,a,b,a+b;
  // maps from r1 to r2,
  // x -> a
  // y -> b
  // z -> a+b
  f(i);
↦ _[1]=a
↦ _[2]=b
↦ _[3]=a+b
  // operations like f(i[1]) or f(i*i) are not allowed
  ideal i=f(i);
  // objects in different rings may have the same name
  map g   = r2,a2,b2;
  map phi = g(f);
  // composition of map f and g
  // maps from r1 to r2,
  // x -> a2
  // y -> b2
  // z -> a2+b2
  phi(i);
↦ _[1]=a2
↦ _[2]=b2
↦ _[3]=a2+b2
```

See Section 5.1.38 [fetch], page 182; Section 4.6.2 [ideal expressions], page 81; Section 5.1.59 [imap], page 198; Section 4.12 [map], page 106; Section 4.20 [ring], page 127.

4.12.2 map expressions

A map expression is:

1. an identifier of type map
2. a function returning map
3. map expressions combined by composition using parentheses ((,))

4.12.3 map operations

`( )` composition of maps. If, for example, `f` and `g` are maps, then `f(g)` is a map expression giving the composition $f \circ g$ of `f` and `g`,
provided the target ring of `g` is the basering of `f`.

map_expression `[` int_expressions `]`
is a map entry (the image of the corresponding variable)

Example:

```
  ring r=0,(x,y),dp;
  map f=r,y,x;    // the map f permutes the variables
  f;
↦ f[1]=y
↦ f[2]=x
  poly p=x+2y3;
  f(p);
↦ 2x3+y
  map g=f(f);    // the map g defined as  f^2 is the identity
  g;
↦ g[1]=x
↦ g[2]=y
  g(p) == p;
↦ 1
```

4.12.4 map related functions

`fetch` the identity map between rings (see Section 5.1.38 [fetch], page 182)

`imap` a convenient map procedure for inclusions and projections of rings (see Section 5.1.59 [imap], page 198)

`preimage` preimage under a ring map (see Section 5.1.117 [preimage], page 240)

`subst` substitution of a ring variable (see Section 5.1.154 [subst], page 274)

See also the libraries Section D.4.2 [algebra_lib], page 1003 and Section D.2.12 [ring_lib], page 942, which contain more functions, related to maps.

4.13 matrix

Objects of type matrix are matrices with polynomial entries. Like polynomials they can only be defined or accessed with respect to a basering. In order to compute with matrices having integer or rational entries, define a ring with characteristic 0 and at least one variable.

A matrix can be multiplied by and added to a poly; in this case the polynomial is converted into a matrix of the right size with the polynomial on the diagonal.

If A is a matrix then the assignment `module M=A;` or `module M=module(A);` creates a module generated by the columns of A. Note that the trailing zero columns of A may be deleted by module operations with M.

4.13.1 matrix declarations

Syntax: `matrix` name`[`rows`]` `[`cols`]` `=` list_of_poly_expressions `;`
`matrix` name = matrix_expression `;`

Purpose: defines a matrix (of polynomials).

The given poly_list fills up the matrix beginning with the first row from the left to the right, then the second row and so on. If the poly_list contains less than rows*cols elements, the matrix is filled up with zeros; if it contains more elements, then only the first rows*cols elements are used. If the right-hand side is a matrix expression the matrix on the left-hand side gets the same size as the right-hand side, otherwise the size is determined by the left-hand side. If the size is omitted a 1x1 matrix is created.

Default: 0 (1 x 1 matrix)

Example:

```
  int ro = 3;
  ring r = 32003,(x,y,z),dp;
  poly f=xyz;
  poly g=z*f;
  ideal i=f,g,g^2;
  matrix m[ro][3] = x3y4, 0, i, f ; // a 3 x 3 matrix
  m;
↦ m[1,1]=x3y4
↦ m[1,2]=0
↦ m[1,3]=xyz
↦ m[2,1]=xyz2
↦ m[2,2]=x2y2z4
↦ m[2,3]=xyz
↦ m[3,1]=0
↦ m[3,2]=0
↦ m[3,3]=0
  print(m);
↦ x3y4,0,     xyz,
↦ xyz2,x2y2z4,xyz,
↦ 0,   0,     0
  matrix A;   // the 1 x 1 zero matrix
  matrix B[2][2] = m[1..2, 2..3]; //defines a submatrix
  print(B);
↦ 0,     xyz,
↦ x2y2z4,xyz
  matrix C=m; // defines C as a 3 x 3 matrix equal to m
  print(C);
↦ x3y4,0,     xyz,
↦ xyz2,x2y2z4,xyz,
↦ 0,   0,     0
```

4.13.2 matrix expressions

A matrix expression is:

1. an identifier of type matrix
2. a function returning matrix
3. matrix expressions combined by the arithmetic operations +, - or *
4. a type cast to matrix (see Section 4.13.3 [matrix type cast], page 110)

Example:

```
  ring r=0,(x,y),dp;
  poly f= x3y2 + 2x2y2 +2;
  matrix H = jacob(jacob(f));    // the Hessian of f
  matrix mc = coef(f,y);
  print(mc);
↦ y2,    1,
↦ x3+2x2,2
  module MD = [x+y,1,x],[x+y,0,y];
  matrix M = MD;
  print(M);
↦ x+y,x+y,
↦ 1,  0,
↦ x,  y
```

4.13.3 matrix type cast

Syntax: `matrix` (expression)
`matrix` (expression, int_n, int_m)

Type: matrix

Purpose: Converts expression to a matrix, where expression must be of type int, intmat, intvec, number, poly, ideal, vector, module, or matrix. If int_n and int_m are supplied, then they specify the dimension of the matrix. Otherwise, the size (resp. dimensions) of the matrix is determined by the size (resp. dimensions) of the expression.

Example:

```
  ring r=32003,(x,y,z),dp;
  matrix(x);
↦ _[1,1]=x
  matrix(x, 1, 2);
↦ _[1,1]=x
↦ _[1,2]=0
  matrix(intmat(intvec(1,2,3,4), 2, 2));
↦ _[1,1]=1
↦ _[1,2]=2
↦ _[2,1]=3
↦ _[2,2]=4
  matrix(_, 2, 3);
↦ _[1,1]=1
↦ _[1,2]=2
↦ _[1,3]=0
↦ _[2,1]=3
↦ _[2,2]=4
↦ _[2,3]=0
  matrix(_, 2, 1);
↦ _[1,1]=1
↦ _[2,1]=3
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.8.3 [intmat type cast], page 92; Section 4.13 [matrix], page 108.

4.13.4 matrix operations

`+` addition with matrix or poly; the polynomial is converted into a diagonal matrix

`-` negation or subtraction with matrix or poly (the first operand is expected to be a matrix); the polynomial is converted into a diagonal matrix

`*` multiplication with matrix or poly; the polynomial is converted into a diagonal matrix

`/` division by poly

`==`, `<>`, `!=` comparators

matrix_expression `[` int_expression`,` int_expression `]`
is a matrix entry, where the first index indicates the row and the second the column

Example:

```
  ring r=32003,x,dp;
  matrix A[3][3] = 1,3,2,5,0,3,2,4,5; // define a matrix
  print(A); // nice printing of small matrices
↦ 1,3,2,
↦ 5,0,3,
↦ 2,4,5
  A[2,3];   // matrix entry
↦ 3
  A[2,3] = A[2,3] + 1; // change entry
  A[2,1..3] = 1,2,3;   // change 2nd row
  print(A);
↦ 1,3,2,
↦ 1,2,3,
↦ 2,4,5
  matrix E[3][3]; E = E + 1;  // the unit matrix
  matrix B =x*E - A;
  print(B);
↦ x-1,-3, -2,
↦ -1, x-2,-3,
↦ -2, -4, x-5
  // the same (but x-A does not work):
  B = -A+x;
  print(B);
↦ x-1,-3, -2,
↦ -1, x-2,-3,
↦ -2, -4, x-5
  det(B);        // the characteristic polynomial of A
↦ x3-8x2-2x-1
  A*A*A - 8 * A*A - 2*A == E;  // Cayley-Hamilton
↦ 1
  vector v =[x,-1,x2];
  A*v; // multiplication of matrix and vector
↦ _[1,1]=2x2+x-3
↦ _[2,1]=3x2+x-2
↦ _[3,1]=5x2+2x-4
  matrix m[2][2]=1,2,3;
  print(m-transpose(m));
↦ 0,-1,
↦ 1,0
```

4.13.5 matrix related functions

`bareiss` Gauss-Bareiss algorithm (see Section 5.1.3 [bareiss], page 158)

`coef` matrix of coefficients and monomials (see Section 5.1.11 [coef], page 164)

`coeffs` matrix of coefficients (see Section 5.1.12 [coeffs], page 165)

`det` determinant (see Section 5.1.23 [det], page 172)

`diff` partial derivative (see Section 5.1.24 [diff], page 173)

`jacob` Jacobi matrix (see Section 5.1.66 [jacob], page 203)

`koszul` Koszul matrix (see Section 5.1.73 [koszul], page 207)

`lift` lift-matrix (see Section 5.1.80 [lift], page 211)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.81 [liftstd], page 212)

`minor` set of minors of a matrix (see Section 5.1.92 [minor], page 221)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232)

`print` nice print format (see Section 5.1.120 [print], page 242)

`size` number of matrix entries (see Section 5.1.144 [size], page 264)

`subst` substitute a ring variable (see Section 5.1.154 [subst], page 274)

`trace` trace of a matrix (see Section 5.1.158 [trace], page 281)

`transpose`
transposed matrix (see Section 5.1.159 [transpose], page 282)

`wedge` wedge product (see Section 5.1.171 [wedge], page 288)

See also the library Section D.3.1 [matrix_lib], page 967, which contains more matrix-related functions.

4.14 module

Modules are submodules of a free module over the basering with basis `gen(1)`, `gen(2)`, ... They are represented by lists of vectors which generate the submodule. Like vectors they can only be defined or accessed with respect to a basering.

If R is the basering, and M is a submodule of R^n

generated by vectors $v_1, \ldots, v_k$, then $v_1, \ldots, v_k$

may be considered as the generators of relations of R^n/M between the canonical generators `gen(1)`,. . .,`gen(n)`. Hence any finitely generated R -module can be represented in SINGULAR by its module of relations. The assignments `module M=v1,...,vk; matrix A=M;` create the presentation matrix of size n × k for R^n/M , i.e., the columns of A are the vectors $v_1, \ldots, v_k$ which generate M (cf. Section B.1 [Representation of mathematical objects], page 767).

4.14.1 module declarations

Syntax: `module` name = list_of_vector_expressions ;
`module` name = module_expression ;

Purpose: defines a module.

Default: [0]

Example:

```
  ring r=0,(x,y,z),(c,dp);
  vector s1 = [x2,y3,z];
  vector s2 = [xy,1,0];
  vector s3 = [0,x2-y2,z];
  poly   f  = xyz;
  module m = s1, s2-s1,f*(s3-s1);
  m;
↦ m[1]=[x2,y3,z]
↦ m[2]=[-x2+xy,-y3+1,-z]
↦ m[3]=[-x3yz,-xy4z+x3yz-xy3z]
  // show m in matrix format (columns generate m)
  print(m);
↦ x2,-x2+xy,-x3yz,
↦ y3,-y3+1, -xy4z+x3yz-xy3z,
↦ z, -z,    0
```

4.14.2 module expressions

A module expression is:

1. an identifier of type module
2. a function returning module
3. module expressions combined by the arithmetic operation `+`
4. multiplication of a module expression with an ideal or a poly expression: `*`
5. a type cast to module

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.6 [ideal], page 80; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

4.14.3 module operations

`+` addition (concatenation of the generators and simplification)

`*` multiplication with ideal or poly (but not 'module' * 'module'!)

module_expression [int_expression , int_expression]
is a module entry, where the first index indicates the row and the second the column

module_expressions [int_expression]
is a vector, where the index indicates the column (generator)

Example:

```
ring r=0,(x,y,z),dp;
module m=[x,y],[0,0,z];
```

```
  print(m*(x+y));
↦ x2+xy,0,
↦ xy+y2,0,
↦ 0,    xz+yz
  // this is not distributive:
  print(m*x+m*y);
↦ x2,0, xy,0,
↦ xy,0, y2,0,
↦ 0, xz,0, yz
```

4.14.4 module related functions

`coeffs` matrix of coefficients (see Section 5.1.12 [coeffs], page 165)

`degree` multiplicity, dimension and codimension of the module of leading terms (see Section 5.1.20 [degree], page 171)

`diff` partial derivative (see Section 5.1.24 [diff], page 173)

`dim` Krull dimension of free module over the basering modulo the module of leading terms (see Section 5.1.25 [dim], page 174)

`eliminate`
elimination of variables (see Section 5.1.28 [eliminate], page 176)

`freemodule`
the free module of given rank (see Section 5.1.47 [freemodule], page 188)

`fres` free resolution of a standard basis (see Section 5.1.48 [fres], page 188)

`groebner` Groebner basis computation (a wrapper around `std,stdhilb,stdfglm`,...) (see Section 5.1.53 [groebner], page 191)

`hilb` Hilbert function of a standard basis (see Section 5.1.56 [hilb], page 195)

`homog` homogenization with respect to a variable (see Section 5.1.57 [homog], page 196)

`interred` interreduction of a module (see Section 5.1.64 [interred], page 201)

`intersect`
module intersection (see Section 5.1.65 [intersect], page 202)

`jet` Taylor series up to a given order (see Section 5.1.68 [jet], page 204)

`kbase` vector space basis of free module over the basering modulo the module of leading terms (see Section 5.1.69 [kbase], page 205)

`lead` initial module (see Section 5.1.75 [lead], page 209)

`lift` lift-matrix (see Section 5.1.80 [lift], page 211)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.81 [liftstd], page 212)

`lres` free resolution (see Section 5.1.83 [lres], page 215)

`minbase` minimal generating set of a homogeneous ideal, resp. module, or an ideal, resp. module, over a local ring

`modulo` represents $(h1 + h2)/h1 = h2/(h1 \cap h2)$ (see Section 5.1.94 [modulo], page 223)

`mres` minimal free resolution of an ideal resp. module w.r.t. a minimal set of generators of the given module (see Section 5.1.98 [mres], page 225) `mult` multiplicity, resp. degree, of the module of leading terms (see Section 5.1.101 [mult], page 228)

`nres` computation of a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 5.1.106 [nres], page 232)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232)

`print` nice print format (see Section 5.1.120 [print], page 242)

`prune` minimization of the embedding into a free module (see Section 5.1.122 [prune], page 245)

`qhweight` quasihomogeneous weights of an ideal, resp. module (see Section 5.1.124 [qhweight], page 246)

`quotient` module quotient (see Section 5.1.127 [quotient], page 248)

`reduce` normalform with respect to a standard basis (see Section 5.1.131 [reduce], page 251)

`res` free resolution of an ideal, resp. module, but not changing the given ideal, resp. module (see Section 5.1.134 [res], page 253)

`simplify` simplification of a set of vectors (see Section 5.1.143 [simplify], page 263)

`size` number of non-zero generators (see Section 5.1.144 [size], page 264)

`sortvec` permutation for sorting ideals/modules (see Section 5.1.146 [sortvec], page 266)

`sres` free resolution of a standard basis (see Section 5.1.149 [sres], page 269)

`std` standard basis computation (see Section 5.1.151 [std], page 271, Section 5.1.81 [liftstd], page 212)

`subst` substitution of a ring variable (see Section 5.1.154 [subst], page 274)

`syz` computation of the first syzygy module (see Section 5.1.156 [syz], page 280)

`vdim` vector space dimension of free module over the basering modulo module of leading terms (see Section 5.1.168 [vdim], page 286)

`weight` "optimal" weights (see Section 5.1.172 [weight], page 288)

4.15 number

Numbers are elements from the coefficient ring (or ground ring). They can only be defined or accessed with respect to a basering which determines the coefficient field. See Section 4.20.2 [ring declarations], page 127 for declarations of coefficient fields.

Warning: Beware of the special meaning of the letter `e` (immediately following a sequence of digits) if the field is real (or complex), Section 6.4 [Miscellaneous oddities], page 313.

4.15.1 number declarations

Syntax: `number` name = number_expression ;

Purpose: defines a number.

Default: 0

Note: Numbers may only be declared w.r.t. the coefficient field of the current basering, i.e., a ring has to be defined prior to any number declaration. See Section 3.3 [Rings and orderings], page 30 for a list of the available coefficient fields.

Example:

```
  // finite field Z/p, p<= 32003
  ring r = 32003,(x,y,z),dp;
  number n = 4/6;
  n;
↦ -10667
  // finite field GF(p^n), p^n <= 32767
  // z is a primitive root of the minimal polynomial
  ring rg= (7^2,z),x,dp;
  number n = 4/9+z;
  n;
↦ z38
  // the rational numbers
  ring r0 = 0,x,dp;
  number n = 4/6;
  n;
↦ 2/3
  // algebraic extensions of Z/p or Q
  ring ra=(0,a),x,dp;
  minpoly=a^2+1;
  number n=a3+a2+2a-1;
  n;
↦ (a-2)
  a^2;
↦ -1
  // transcedental extensions of Z/p or Q
  ring rt=(0,a),x,dp;
  number n=a3+a2+2a-1;
  n;
↦ (a3+a2+2a-1)
  a^2;
↦ (a2)
  // machine floating point numbers, single precision
  ring R_0=real,x,dp;
  number n=4/6;
  n;
↦ (6.667e-01)
  n=0.25e+2;
  n;
↦ (2.500e+01)
  // floating point numbers, arbitrary prescribed precision
  ring R_1=(real,50),x,dp;
  number n=4.0/6;
  n;
↦ 0.66666666666666666666666666666666666666666666666667
  n=0.25e+2;
  n;
↦ 25
  // floating point complex numbers, arbitrary prescribed precision
  // the third parameter gives the name of the imaginary unit
  ring R_2=(complex,50,i),x,dp;
  number n=4.0/6;
  n;
↦ 0.66666666666666666666666666666666666666666666666667
  n=0.25e+2*i+n;
  n;
↦ (0.66666666666666666666666666666666666666666666666667+i*25)
```

4.15.2 number expressions

A number expression is:

1. a rational number (there are NO spaces allowed inside a rational number, see Section 4.7.2 [int expressions], page 85)
2. a floating point number (if the coefficient field is `real`):
   <digits>.<digits>e<sign><digits>
3. an identifier of type number
4. a function returning number
5. an int expression (see Section 3.5.5 [Type conversion and casting], page 46)
6. number expressions combined by the arithmetic operations `+`, `-`, `*`, `/`, `^`, or `**`.
7. a type cast to number

Example:

```
  // the following expressions are in any ring int expressions
  2 / 3;
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  2 / 3;<<
↦ 0
  4/ 8;
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  4/ 8;<<
↦ 0
  2 /2;   // the notation of / for div might change in the future
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  2 /2;   // the\
    notation of / for div might change in the future<<
↦ 1
  ring r0=0,x,dp;
  2/3, 4/8, 2/2 ; // are numbers
↦ 2/3 1/2 1

  poly f = 2x2 +1;
  leadcoef(f);
↦ 2
  typeof(_);
↦ number
  ring rr =real,x,dp;
  1.7e-2; 1.7e+2; // are valid (but  1.7e2 not), if the field is ‘real‘
↦ (1.700e-02)
↦ (1.700e+02)
  ring rp = (31,t),x,dp;
  2/3, 4/8, 2/2 ; // are numbers
↦ 11 -15 1
```

```
  poly g = (3t2 +1)*x2 +1;
  leadcoef(g);
↦ (3t2+1)
  typeof(_);
↦ number
  par(1);
↦ (t)
  typeof(_);
↦ number
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.20 [ring], page 127.

4.15.3 number operations

`+`	addition
`-`	negation or subtraction
`*`	multiplication
`/`	division
`%`, `mod`	modulo
`^`, `**`	power, exponentiation (by an integer)
`<=`, `>=`, `==`, `<>`	comparison
`mod`	integer modulo (the remainder of the division `div`), always non-negative

Note: Quotient and exponentiation is only recognized as a number expression if it is already a number, see Section 6.4 [Miscellaneous oddities], page 313.
For the behavior of comparison operators in rings with ground field different from real or the rational numbers, see Section 4.7.5 [boolean expressions], page 89.

Example:

```
  ring r=0,x,dp;
  number n = 1/2 +1/3;
  n;
↦ 5/6
  n/2;
↦ 5/12
  1/2/3;
↦ 1/6
  1/2 * 1/3;
↦ 1/6
  n = 2;
  n^-2;
↦ 1/4
  // the following oddities appear here
  2/(2+3);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  2/(2+3);<<
↦ 0
  number(2)/(2+3);
↦ 2/5
  2^-2; // for int’s exponent must be non-negative
```

```
↦    ? exponent must be non-negative
↦    ? error occurred in or before ./examples/number_operations.sing line 1\
   2: ‘  2^-2; // for int’s exponent must be non-negative‘
  number(2)^-2;
↦ 1/4
  3/4>=2/5;
↦ 1
  2/6==1/3;
↦ 1
```

4.15.4 number related functions

`cleardenom`
: cancellation of denominators of numbers in polyomial and divide it by its content (see Section 5.1.9 [cleardenom], page 163)

`impart`
: imaginary part of a complex number, 0 otherwise (see Section 5.1.60 [impart], page 199, Section 5.1.133 [repart], page 253)

`numerator, denominator`
: the numerator/denominator of a rational number (see Section 5.1.108 [numerator], page 233, Section 5.1.22 [denominator], page 172)

`leadcoef`
: coefficient of the leading term (see Section 5.1.76 [leadcoef], page 209)

`par`
: n-th parameter of the basering (see Section 5.1.114 [par], page 239)

`pardeg`
: degree of a number in ring parameters (see Section 5.1.115 [pardeg], page 239)

`parstr`
: string form of ring parameters (see Section 5.1.116 [parstr], page 240)

`repart`
: real part of a complex number (see Section 5.1.60 [impart], page 199, Section 5.1.133 [repart], page 253)

4.16 package

The data type package is used to group identifiers into collections. It is mainly used as an internal means to avoid collisions of names of identifiers in libraries with variable names defined by the user. The most important package is the toplevel package, called `Top`. It contains all user defined identifiers as well as all user accessible library procedures. Identifiers which are local to a library are contained in a package whose name is obtained from the name of the library, where the first letter is converted to uppercase, the remaining ones to lowercase. Another reserved package name is `Current` which denotes the current package name in use. See also Section 3.8 [Libraries], page 55.

4.16.1 package declarations

Syntax: `package` name ;

Purpose: defines a package (Only relevant in very special situations).

Example:

```
  package Test;
  int i=3; exportto(Test,i);
  Test::i+2;
↦ 5
  i;
```

```
↦     ? ‘i‘ is undefined
↦     ? error occurred in or before ./examples/package_declarations.sing li
   e 4: ‘  i;‘
  listvar();
  listvar(Test);
↦ // Test                            [0]  package Test (N)
↦ // ::i                             [0]  int 3
  package dummy = Test;
  kill Test;
  listvar(dummy);
↦ // dummy                           [0]  package dummy (N)
↦ // ::i                             [0]  int 3
```

4.16.2 package related functions

`exportto` transfer an identifier to the specified package (see Section 5.2.7 [exportto], page 293)

`importfrom`
generate a copy of an identifier from the specified package in the current package (see Section 5.2.10 [importfrom], page 296)

`listvar` list variables currently defined in a given package (see Section 5.1.82 [listvar], page 213)

`load` load a library or dynamic module (see Section 5.2.12 [load], page 299)

`LIB` load a library or dynamic module (see Section 5.1.79 [LIB], page 211)

4.17 poly

Polynomials are the basic data for all main algorithms in Singular. They consist of finitely many terms (coefficient*monomial) which are combined by the usual polynomial operations (see Section 4.17.2 [poly expressions], page 121). Polynomials can only be defined or accessed with respect to a basering which determines the coefficient type, the names of the indeterminates and the monomial ordering.

```
ring r=32003,(x,y,z),dp;
poly f=x3+y5+z2;
```

4.17.1 poly declarations

Syntax: `poly` name `=` poly_expression `;`

Purpose: defines a polynomial.

Default: 0

Example:

```
  ring r = 32003,(x,y,z),dp;
  poly s1  = x3y2+151x5y+186xy6+169y9;
  poly s2  = 1*x^2*y^2*z^2+3z8;
  poly s3  = 5/4x4y2+4/5*x*y^5+2x2y2z3+y7+11x10;
  int a,b,c,t=37,5,4,1;
  poly f=3*x^a+x*y^(b+c)+t*x^a*y^b*z^c;
  f;
↦ x37y5z4+3x37+xy9
  short = 0;
```

```
                  f;
                ↦ x^37*y^5*z^4+3*x^37+x*y^9
```

Section 5.3.7 [short], page 305

4.17.2 poly expressions

A polynomial expression is (optional parts in square brackets):

1. a monomial (there are NO spaces allowed inside a monomial)

   `[coefficient] ring_variable [ exponent] [ring_variable [exponent] ...].`

   Monomials which contain an indexed ring variable must be built from `ring_variable` and `coefficient` with the operations `*` and `^`
2. an identifier of type poly
3. a function returning poly
4. polynomial expressions combined by the arithmetic operations `+`, `-`, `*`, `/`, or `^`
5. an int expression (see Section 3.5.5 [Type conversion and casting], page 46)
6. a type cast to poly

Example:

```
ring S=0,(x,y,z,a(1)),dp;
2x, x3, 2x2y3, xyz, 2xy2; //  are monomials
2*x, x^3, 2*x^2*y^3, x*y*z, 2*x*y^2; // are poly expressions
2*a(1); // is a valid polynomial expression (a(1) is a name of a variable),
        // but not 2a(1) (is a syntax error)
2*x^3;  // is a valid polynomial expression equal to 2x3 (a valid monomial)
        // but not equal to 2x^3 which will be interpreted as (2x)^3
        // since 2x is a monomial
  ring r=0,(x,y),dp;
  poly f = 10x2y3 +2x2y2-2xy+y -x+2;
  lead(f);
↦ 10x2y3
  leadmonom(f);
↦ x2y3
  simplify(f,1);     // normalize leading coefficient
↦ x2y3+1/5x2y2-1/5xy-1/10x+1/10y+1/5
  poly g = 1/2x2 + 1/3y;
  cleardenom(g);
↦ 3x2+2y
  int i = 102;
  poly(i);
↦ 102
  typeof(_);
↦ poly
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.20 [ring], page 127.

4.17.3 poly operations

`+` addition

`-` negation or subtraction

`*` multiplication

`/, div` division by a polynomial, ignoring the remainder (only implemented for polynomials over QQ, ZZ/p and field extensions of them)
(See also Section 5.1.127 [quotient], page 248,Section 5.1.26 [division], page 174,Section 5.1.131 [reduce], page 251)

`%, mod` the remainder from the division by a polynomial (only implemented for polynomials over QQ, ZZ/p and field extensions of them)
(See also Section 5.1.127 [quotient], page 248,Section 5.1.26 [division], page 174,Section 5.1.131 [reduce], page 251)

`^, **` power by a positive integer

`<, <=, >, >=, ==, <>`
comparators (considering leading monomials w.r.t. monomial ordering)

poly_expression `[` intvec_expression `]`
the sum of monomials at the indicated places w.r.t. the monomial ordering

Example:

```
  ring R=0,(x,y),dp;
  poly f = x3y2 + 2x2y2 + xy - x + y + 1;
  f;
↦ x3y2+2x2y2+xy-x+y+1
  f + x5 + 2;
↦ x5+x3y2+2x2y2+xy-x+y+3
  f * x2;
↦ x5y2+2x4y2+x3y-x3+x2y+x2
  (x+y)/x;
↦ 1
  f/3x2;
↦ 1/3xy2+2/3y2
  x5 > f;
↦ 1
  x<=y;
↦ 0
  x>y;
↦ 1
  ring r=0,(x,y),ds;
  poly f = fetch(R,f);
  f;
↦ 1-x+y+xy+2x2y2+x3y2
  x5 > f;
↦ 0
  f[2..4];
↦ -x+y+xy
  size(f);
↦ 6
  f[size(f)+1]; f[-1];    // monomials out of range are 0
↦ 0
↦ 0
  intvec v = 6,1,3;
  f[v];          // the polynom built from the 1st, 3rd and 6th monomial of f
↦ 1+y+x3y2
```

4.17.4 poly related functions

`cleardenom`
cancellation of denominators of numbers in polynomial and divide it by its content (see Section 5.1.9 [cleardenom], page 163; Section D.2.8.14 [content], page 894)

`coef` matrix of coefficients and monomials (see Section 5.1.11 [coef], page 164)

`coeffs` matrix of coefficients (see Section 5.1.12 [coeffs], page 165)

`deg` degree (see Section 5.1.19 [deg], page 170)

`diff` partial derivative (see Section 5.1.24 [diff], page 173)

`extgcd` Bezout representation of gcd (see Section 5.1.33 [extgcd], page 178)

`factorize`
factorization of polynomial (see Section 5.1.36 [factorize], page 180)

`finduni` univariate polynomials in a zero-dimensional ideal (see Section 5.1.43 [finduni], page 185)

`gcd` greatest common divisor (see Section 5.1.50 [gcd], page 190)

`homog` homogenization (see Section 5.1.57 [homog], page 196)

`jacob` ideal, resp. matrix, of all partial derivatives (see Section 5.1.66 [jacob], page 203)

`lead` leading term (see Section 5.1.75 [lead], page 209)

`leadcoef` coefficient of the leading term (see Section 5.1.76 [leadcoef], page 209)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.77 [leadexp], page 210)

`leadmonom`
leading monomial (see Section 5.1.78 [leadmonom], page 210)

`jet` monomials of degree at most k (see Section 5.1.68 [jet], page 204)

`ord` degree of the leading monomial (see Section 5.1.112 [ord], page 238)

`qhweight` quasihomogeneous weights (see Section 5.1.124 [qhweight], page 246)

`reduce` normal form with respect to a standard base (see Section 5.1.131 [reduce], page 251)

`rvar` test for ring variable (see Section 5.1.139 [rvar], page 258)

`simplify` normalization of a polynomial (see Section 5.1.143 [simplify], page 263)

`size` number of monomials (see Section 5.1.144 [size], page 264)

`subst` substitution of a ring variable (see Section 5.1.154 [subst], page 274)

`trace` trace of a matrix (see Section 5.1.158 [trace], page 281)

`var` the indicated variable of the ring (see Section 5.1.165 [var], page 285)

`varstr` variable(s) in string form (see Section 5.1.167 [varstr], page 285)

4.18 proc

Procedures are sequences of SINGULAR commands in a special format. They are used to extend the set of SINGULAR commands with user defined commands. Once a procedure is defined it can be used as any other SINGULAR command. Procedures may be defined by either typing them on the command line or by loading them from a file. For a detailed description on the concept of procedures in SINGULAR see Section 3.7 [Procedures], page 50. A file containing procedure definitions which comply with certain syntax rules is called a library. Such a file is loaded using the command `LIB`. For more information on libraries see Section 3.8 [Libraries], page 55.

4.18.1 proc declaration

Syntax:

```
[static] proc proc_name [(<parameter_list>)
[<help_string>]
{
  <procedure_body>
}
[example
{
  <sequence_of_commands>
}]
```

Purpose: Defines a new function, the `proc` proc_name. Once loaded in a SINGULAR session, the information provided in the help string will be displayed upon entering `help proc_name;`, while the `example` section will be executed upon entering `example proc_name;`. See Section 3.7.2 [Parameter list], page 52, Section 3.7.3 [Help string], page 54, and the example in Section 3.8.6 [Procedures in a library], page 57.

The help string, the parameter list, and the example section are optional. They are, however, mandatory for the procedures listed in the header of a library. The help string is ignored and no example section is allowed if the procedure is defined interactively, i.e., if it is not loaded from a file by the `LIB` or `load` command (see Section 5.1.79 [LIB], page 211 and see Section 5.2.12 [load], page 299).

In the body of a library, each procedure not meant to be accessible by users should be declared static. See Section 3.8.6 [Procedures in a library], page 57.

Example:

```
  proc milnor_number (poly p)
  {
    ideal i= std(jacob(p));
    int m_nr=vdim(i);
    if (m_nr<0)
    {
      "// not an isolated singularity";
    }
    return(m_nr);         // the value of m_nr is returned
  }
  ring r1=0,(x,y,z),ds;
  poly p=x^2+y^2+z^5;
  milnor_number(p);
↦ 4
```

See Section 5.1.79 [LIB], page 211; Section 3.8 [Libraries], page 55; Section 5.2.1 [apply], page 290.

4.18.2 proc expression

Syntax: variable_name `->` { expression(s) }

Purpose: Defines a new function, within `apply` or for assigning.

Example:

```
  apply(1..3,x->{x**2});
↦ 1 4 9
```

See Section 5.2.1 [apply], page 290; Section 4.18 [proc], page 124.

4.18.3 procs with different argument types

Syntax: `branchTo (` string_expression `,` ... proc_name `)`

Purpose: branch to the given procedure if the argument types matches the types given as strings (which may be empty - matching the empty argument list). The main procedure (`p` in the example) must be defined without an argument list, and `branchTo` statement must be the first statement within the procedure body.

Example:

```
  proc p1(int i) { "int:",i; }
  proc p21(string s) { "string:",s; }
  proc p22(string s1, string s2) { "two strings:",s1,s2; }
  proc p()
  { branchTo("int",p1);
    branchTo("string","string",p22);
    branchTo("string",p21);
    ERROR("not defined for these argument types");
  }
  p(1);
↦ int: 1
  p("hu");
↦ string: hu
  p("ha","ha");
↦ two strings: ha ha
  p(1,"hu");
↦    ? not defined for these argument types
↦    ? leaving ::p (0)
```

See Section 4.18 [proc], page 124.

4.19 resolution

The type resolution is intended as an intermediate representation which internally retains additional information obtained during computation of resolutions. It furthermore enables the use of partial results to compute, for example, Betti numbers or minimal resolutions. Like ideals and modules, a resolution can only be defined w.r.t. a basering (see Section C.3 [Syzygies and resolutions], page 775).

Note: To access the elements of a resolution, it has to be assigned to a list. This assignment also completes computations and may therefore take time, (resp. an access directly with the brackets `[ , ]` causes implicitly a cast to a list).

4.19.1 resolution declarations

Syntax: `resolution` name = resolution_expression ;

Purpose: defines a resolution.

Default: none

Example:

```
  ring R;
  ideal i=z2,x;
  resolution re=res(i,0);
  re;
↦  1      2      1
↦ R <--  R <--  R
↦
↦ 0      1      2
↦
  betti(re);
↦ 1,1,0,
↦ 0,1,1
  list l = re;
  l;
↦ [1]:
↦    _[1]=x
↦    _[2]=z2
↦ [2]:
↦    _[1]=-z2*gen(1)+x*gen(2)
↦ [3]:
↦    _[1]=0
```

4.19.2 resolution expressions

A resolution expression is:

1. an identifier of type resolution
2. a function returning a resolution
3. a type cast to resolution from a list of ideals, resp. modules..

See Section 3.5.5 [Type conversion and casting], page 46.

4.19.3 resolution related functions

`betti` Betti numbers of a resolution (see Section 5.1.4 [betti], page 159)

`fres` free resolution of a standard basis (see Section 5.1.48 [fres], page 188)

`lres` free resolution (see Section 5.1.83 [lres], page 215)

`minres` minimize a free resolution (see Section 5.1.93 [minres], page 223)

`mres` minimal free resolution of an ideal, resp. module and a minimal set of generators of the given ideal, resp. module (see Section 5.1.98 [mres], page 225)

`res` free resolution of an ideal, resp. module, but not changing the given ideal, resp. module (see Section 5.1.134 [res], page 253)

`sres` free resolution of a standard basis (see Section 5.1.149 [sres], page 269)

4.20 ring

Rings are used to describe properties of polynomials, ideals etc. Almost all computations in Singular require a basering. For a detailed description of the concept of rings see Section 3.3 [Rings and orderings], page 30.

4.20.1 qring

Singular offers the opportunity to calculate in quotient rings (factor rings), i.e., rings modulo an ideal. The ideal has to be given as a standard basis. For a detailed description of the concept of rings and quotient rings see Section 3.3 [Rings and orderings], page 30. Beside the construction, an object describing a quotient ring is of type `ring`.

See Section 4.20.5 [qring declaration], page 129.

4.20.2 ring declarations

Syntax: `ring` name `=` `(` coefficients `)`, `(` names_of_ring_variables `)`, `(` ordering `);` or
`ring` name `=` cring `[` names_of_ring_variables `]`

Default: `(ZZ/32003)[x,y,z]`

Purpose: declares a ring and sets it as the actual basering. The second form sets the ordering to `(dp,C)`.

For the second form: `cring` stands currently for `QQ` (the rationals), `ZZ` (the integers) or `(ZZ/m)` (the field (m prime and <2147483648) resp. ring of the integers modulo m).

The coefficients for the first form are given by one of the following:

1. a `cring` as given above
2. a non-negative int_expression less or equal 2147483647.
3. an expression_list of an int_expression and one or more names.
4. the name `real`
5. an expression_list of the name `real` and an int_expression.
6. an expression_list of the name `complex`, an optional int_expression and a name.
7. an expression_list of the name `ZZ`.
8. an expression_list of the name `integer` and following int_expressions.
9. an expression_list of the name `integer` and two int_expressions.

For the definition of the 'coefficients', see Section 3.3 [Rings and orderings], page 30.

'names_of_ring_variables' must be a list of names or (multi-)indexed names.

'ordering' is a list of block orderings where each block ordering is either

1. `lp`, `dp`, `Dp`, `rp`, `ls`, `ds`, `Ds`, or `rs` optionally followed by a size parameter in parentheses.
2. `wp`, `Wp`, `ws`, `Ws`, `am`, `aa`, or `a` followed by a weight vector given as an intvec_expression in parentheses.
3. `M` followed by an intmat_expression in parentheses.
4. `c` or `C`.

For the definition of the orderings, see Section 3.3.3 [Term orderings], page 34, Section B.2 [Monomial orderings], page 768.

If one of coefficients, names_of_ring_variables, and ordering consists of only one entry, the parentheses around this entry may be omitted.

See also Section 3.3.1 [Examples of ring declarations], page 31; Section 4.20 [ring], page 127; Section 5.1.137 [ringlist], page 255.

4.20.3 ring related functions

`charstr` description of the coefficient field of a ring (see Section 5.1.7 [charstr], page 162)

`keepring` move ring to next upper level (see Section 5.2.11 [keepring], page 298)

`npars` number of ring parameters (see Section 5.1.105 [npars], page 231)

`nvars` number of ring variables (see Section 5.1.109 [nvars], page 233)

`ordstr` monomial ordering of a ring (see Section 5.1.113 [ordstr], page 239)

`parstr` names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.116 [parstr], page 240)

`qring` quotient ring (see Section 4.20.1 [qring], page 127)

`ringlist` decomposition of a ring into a list of its components (see Section 5.1.137 [ringlist], page 255)

`setring` setting of a new basering (see Section 5.1.141 [setring], page 260)

`varstr` names of all ring variables or the name of the n-th ring variable (see Section 5.1.167 [varstr], page 285)

4.20.4 ring operations

`+` construct a new ring $k[X,Y]$ from $k_1[X]$ and $k_2[Y]$. (The sets of variables must be distinct).

`==,<>` compare two rings

Note: Concerning the ground fields k_1 and k_2 take the following guide lines into consideration:

- Neither k_1 nor k_2 may be R or C .
- If the characteristic of k_1 and k_2 differs, then one of them must be Q .
- At most one of k_1 and k_2 may have parameters.
- If one of k_1 and k_2 is an algebraic extension of Z/p it may not be defined by a `charstr` of type `(p^n,a)`.

Example:

```
    ring R1=0,(x,y),dp;
    ring R2=32003,(a,b),dp;
    def R=R1+R2;
    R;
  ↦ // coefficients: ZZ/32003
  ↦ // number of vars : 4
  ↦ //        block   1 : ordering dp
  ↦ //                  : names    x y
  ↦ //        block   2 : ordering dp
  ↦ //                  : names    a b
  ↦ //        block   3 : ordering C
```

Section D.2.12 [ring_lib], page 942

4.20.5 qring declaration

Syntax: `qring` name = ideal_expression ;

Default: none

Purpose: declares a quotient ring as the basering modulo ideal_expression and sets it as current basering.

Operations based on standard bases (e.g. `std`,`groebner`, etc., `reduce`) and functions which require a standard basis (e.g. `dim`,`hilb`, etc.) operated with the residue classes; all others on the polynomial objects.

Example:

```
ring r=0,(x,y,z),dp;
ideal i=xy;
qring q=std(i);
basering;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=xy
// simplification is not immediate:
(x+y)^2;
↦ x2+2xy+y2
reduce(_,std(0));
↦ x2+y2
// polynomial and residue class:
ring R=0,(x,y),dp;
qring Q=std(y);
poly p1=x;
poly p2=x+y;
// comparing polynomial objects:
p1==p2;
↦ 0
// comparing residue classes:
reduce(p1,std(0))==reduce(p2,std(0));
↦ 1
```

4.21 smatrix

An experimental type:
Objects of type smatrix are (sparse) matrices with polynomial entries. Like polynomials they can only be defined or accessed with respect to a basering.

Objects of type `smatrix` can be converted to and from `matrix` and `module`.
Operations are `+`, `-`, `*`, `==`, `<>`.
Functions are `chinrem`, `farey`, `ncols`, `nrows`, `std`, `transpose`, `tensor`. Additional `flatten(m)` and `system("unflatten",m,col)`.
Resizing can be done via `smatrix(m,r,c)` where m is of type `module` or `smatrix`.
Access to single entries: `m[i,j]`

See Section 5.1.8 [chinrem], page 162; Section 5.1.37 [farey], page 181; Section D.3.1.5 [flatten], page 970; Section 4.13 [matrix], page 108; Section 4.14 [module], page 112; Section 5.1.104 [ncols], page 231; Section 5.1.107 [nrows], page 232; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271; Section 5.1.157 [tensor], page 281; Section 5.1.159 [transpose], page 282.

4.22 string

Variables of type `string` are used for output (almost every type can be "converted" to `string`) and for creating new commands at runtime see Section 5.1.32 [execute], page 178. They are also return values of certain interpreter related functions (see Section 5.1 [Functions], page 156). String constants consist of a sequence of ANY characters (including newline!) between a starting `"` and a closing `"`. There is also a string constant `newline`, which is the newline character. The `+` sign `"adds"` strings, `""` is the empty string (hence strings form a semigroup). Strings may be used to comment the output of a computation or to give it a nice format. Strings may also be used for intermediate conversion of one type into another.

```
  string s="Hi";
  string s1="a string with new line at the end"+newline;
  string s2="another string with new line at the end
  ";
  s;s1;s2;
↦ Hi
↦ a string with new line at the end
↦
↦ another string with new line at the end
↦
  ring r; ideal i=std(ideal(x,y^3));
  "dimension of i =",dim(i),", multiplicity of i =",mult(i);
↦ dimension of i = 1 , multiplicity of i = 3
  "dimension of i = "+string(dim(i))+", multiplicity of i = "+string(mult(i));
↦ dimension of i = 1, multiplicity of i = 3
  "a"+"b","c";
↦ ab c
```

A comma between two strings makes an expression list out of them (such a list is printed with a separating blank in between), while a `+` concatenates strings.

4.22.1 string declarations

Syntax: `string` name = string_expression ;
`string` name = list_of_string_expressions ;

Purpose: defines a string variable.

Default: `""` (the empty string)

Example:

```
  string s1="Now I know";
  string s2="how to encode a \" in a string...";
  string s=s1+" "+s2; // concatenation of 3 strings
  s;
↦ Now I know how to encode a " in a string...
  s1,s2;   // 2 strings, separated by a blank in the output:
↦ Now I know how to encode a " in a string...
```

4.22.2 string expressions

A string expression is:

1. a sequence of characters between two unescaped quotes (")
2. an identifier of type string
3. a function returning string
4. a substring (using the bracket operator)
5. a type cast to string (see Section 4.22.3 [string type cast], page 131)
6. string expressions combined by the operation +.

Example:

```
// string_expression[start, length] : a substring
// (possibly filled up with blanks)
// the substring of s starting at position 2
// with a length of 4
string s="123456";
s[2,4];
↦ 2345
"abcd"[2,2];
↦ bc
// string_expression[position] : a character from a string
s[3];
↦ 3
// string_expression[position..position] :
// a substring starting at the first position up to the second
// given position
s[2..4];
↦ 2 3 4
// a function returning a string
typeof(s);
↦ string
```

See Section 3.5.5 [Type conversion and casting], page 46

4.22.3 string type cast

Syntax: `string` (expression [, expression_2, ... expression_n])

Type: string

Purpose: Converts each expression to a string, where expression can be of any type. The concatenated string of all converted expressions is returned.

The elements of intvec, intmat, ideal, module, matrix, and list, are separated by a comma. No newlines are inserted.
Not defined elements of a list are omitted.
For link, the name of the link is used.
For map, the ideal defining the mapping is converted.

Note: When applied to a list, elements of type intvec, intmat, ideal, module, matrix, and list become indistinguishable.

Example:

```
  string("1+1=", 2);
↦ 1+1=2
  string(intvec(1,2,3,4));
↦ 1,2,3,4
  string(intmat(intvec(1,2,3,4), 2, 2));
↦ 1,2,3,4
  ring r;
  string(r);
↦ (ZZ/32003),(x,y,z),(dp(3),C)
  string(ideal(x,y));
↦ x,y
  qring R = std(ideal(x,y));
  string(R);
↦ (ZZ/32003),(x,y,z),(dp(3),C)
  map phi = r, ideal(x,z);
  string(phi);
↦ x,z
  list l;
  string(l);
↦
  l[3] = 1;
  string(l); // notice that l[1],l[2] are omitted
↦ 1
  l[2] = l;
  l;
↦ [2]:
↦    [3]:
↦       1
↦ [3]:
↦    1
  string(l); // notice that lists of list is flattened
↦ 1,1
  l[1] = intvec(1,2,3);
  l;
↦ [1]:
↦    1,2,3
↦ [2]:
↦    [3]:
↦       1
↦ [3]:
↦    1
  string(l); // notice that intvec elements are not distinguishable
↦ 1,2,3,1,1
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 5.1.120 [print], page 242; Section 4.22 [string], page 130.

4.22.4 string operations

`+` concatenation

`<=`, `>=`, `==`, `<>`
comparison (lexicographical with respect to the ASCII encoding)

string_expression [int_expression]
: is a character of the string; the index 1 gives the first character.

string_expression [int_expression, int_expression]
: is a substring, where the first argument is the start index and the second is the length of the substring, filled up with blanks if the length exceeds the total size of the string

string_expression [intvec_expression]
: is a expression list of characters from the string

Example:

```
  string s="abcde";
  s[2];
↦ b
  s[3,2];
↦ cd
  ">>"+s[1,10]+"<<";
↦ >>abcde     <<
  s[2]="BC"; s;
↦ aBcde
  intvec v=1,3,5;
  s=s[v]; s;
↦ ace
  s="654321"; s=s[3..5]; s;
↦ 432
```

4.22.5 string related functions

`charstr` description of the coefficient field of a ring (see Section 5.1.7 [charstr], page 162)

`execute` executing string as command (see Section 5.1.32 [execute], page 178)

`find` position of a substring in a string (see Section 5.1.42 [find], page 185)

`names` list of strings of all user-defined variable names (see Section 5.1.103 [names], page 229)

`nameof` name of an object (see Section 5.1.102 [nameof], page 228)

`option` lists all defined options (see Section 5.1.111 [option], page 234)

`ordstr` monomial ordering of a ring (see Section 5.1.113 [ordstr], page 239)

`parstr` names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.116 [parstr], page 240)

`read` read a file (see Section 5.1.130 [read], page 250)

`size` length of a string (see Section 5.1.144 [size], page 264)

`sprintf` string formatting (see Section 5.1.148 [sprintf], page 267)

`typeof` type of an object (see Section 5.1.161 [typeof], page 282)

`varstr` names of all ring variables or the name of the n-th ring variable (see Section 5.1.167 [varstr], page 285)

4.23 vector

Vectors are elements of a free module over the basering with basis `gen(1)`, `gen(2)`, ... Like polynomials they can only be defined or accessed with respect to the basering. Each vector belongs to a free module of rank equal to the biggest index of a generator with non-zero coefficient. Since generators with zero coefficients need not be written any vector may be considered also as an element of a free module of higher rank. (E.g., if `f` and `g` are polynomials then `f*gen(1)+g*gen(3)+gen(4)` may also be written as `[f,0,g,1]` or as `[f,0,g,1,0]`.) Note that the elements of a vector have to be surrounded by square brackets (`[` , `]`) (cf. Section B.1 [Representation of mathematical objects], page 767).

4.23.1 vector declarations

Syntax: `vector` name = vector_expression `;`

Purpose: defines a vector of polynomials (an element of a free module).

Default: [0]

Example:

```
  ring r=0,(x,y,z),(c,dp);
  poly s1 = x2;
  poly s2 = y3;
  poly s3 = z;
  vector v = [s1, s2-s1, s3-s1]+ s1*gen(5);
  // v is a vector in the free module of rank 5
  v;
↦ [x2,y3-x2,-x2+z,0,x2]
```

4.23.2 vector expressions

A vector expression is:

1. an identifier of type vector
2. a function returning vector
3. a polynomial expression (via the canonical embedding `p` ↦ `p*gen(1)`)
4. vector expressions combined by the arithmetic operations `+` or `-`
5. a polynomial expression and a vector expression combined by the arithmetic operation `*`
6. a type cast to vector using the brackets `[` , `]`

Example:

```
  // ordering gives priority to components:
  ring rr=0,(x,y,z),(c,dp);
  vector v=[x2+y3,2,0,x*y]+gen(6)*x6;
  v;
↦ [y3+x2,2,0,xy,0,x6]
  vector w=[z3-x,3y];
  v-w;
↦ [y3-z3+x2+x,-3y+2,0,xy,0,x6]
  v*(z+x);
↦ [xy3+y3z+x3+x2z,2x+2z,0,x2y+xyz,0,x7+x6z]
  // ordering gives priority to monomials:
  // this results in a different output
  ring r=0,(x,y,z),(dp,c);
  imap(rr,v);
↦ x6*gen(6)+y3*gen(1)+x2*gen(1)+xy*gen(4)+2*gen(2)
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 4.20 [ring], page 127.

4.23.3 vector operations

`+` addition

`-` negation or subtraction

`/` division by a monomial, not divisible terms yield 0

`<`, `<=`, `>`, `>=`, `==`, `<>`
comparators (considering leading terms w.r.t. monomial ordering)

vector_expression `[` int_expressions `]`
is a vector entry; the index 1 gives the first entry.

Example:

```
       ring R=0,(x,y),(c,dp);
       [x,y]-[1,x];
     ↦ [x-1,-x+y]
       [1,2,x,4][3];
     ↦ x
```

4.23.4 vector related functions

`cleardenom`
quotient of a vector by its content (see Section 5.1.9 [cleardenom], page 163)

`coeffs` matrix of coefficients (see Section 5.1.12 [coeffs], page 165)

`deg` degree (see Section 5.1.19 [deg], page 170)

`diff` partial derivative (see Section 5.1.24 [diff], page 173)

`gen` i-th generator (see Section 5.1.51 [gen], page 191)

`homog` homogenization (see Section 5.1.57 [homog], page 196)

`jet` k-jet: monomials of degree at most k (see Section 5.1.68 [jet], page 204)

`lead` leading term (see Section 5.1.75 [lead], page 209)

`leadcoef` leading coefficient (see Section 5.1.76 [leadcoef], page 209)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.77 [leadexp], page 210)

`leadmonom`
leading monomial (see Section 5.1.78 [leadmonom], page 210)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232)

`ord` degree of the leading monomial (see Section 5.1.112 [ord], page 238)

`reduce` normal form with respect to a standard base (see Section 5.1.131 [reduce], page 251)

`simplify` normalize a vector (see Section 5.1.143 [simplify], page 263)

`size` number of monomials (see Section 5.1.144 [size], page 264)

`subst` substitute a ring variable (see Section 5.1.154 [subst], page 274)

4.24 User defined types

User defined types are (non-empty) lists with a fixed size whose element can be accessed by names (and not indices). These elements have a predefined type (which can also be a user defined type). If these elements depend on a ring they can only be accessed if their base ring is the current base ring. In contrast to usual lists the elements of a user defined type may belong to different rings.

4.24.1 Definition of a user defined type

Syntax: `newstruct(` name , string_expression `);`
`newstruct(` name , name , string_expression `);`

Purpose: defines a new type with elements given by the last argument (string_expression). The name of the new type is the first argument (of type string) and must be longer than one character.
The second name (of type string) is an already defined type which should be extended by the new type.
The last argument (of type string) must be an comma separated list of a type followed by a name. If there are duplicate member names, the last one wins.
(User defined) member names are restricted to alphanumeric characters and must start with a letter.

Operations:

the only operations of user defined types are:

assignment (between objects of the same or extended type)

`typeof`

`string` and printing

operator `.` to access the elements

Example:

```
  newstruct("nt","int a,poly b,string c");
  nt A;
  nt B;
  A.a=3;
  A.c=string(A.a);
  B=A;
  newstruct("t2","nt","string c");
  t2 C; C.c="t2-c";
  A=C;
  typeof(A);
↦ t2
  A;
↦ c=t2-c
↦ c=
↦ b=??
↦ a=0
  // a motivating example ------------------------------------------
  newstruct("IDEAL","ideal I,proc prettyprint");
  newstruct("HOMOGENEOUS_IDEAL","IDEAL","intvec weights,proc prettyprint")
  proc IDEAL_pretty_print(IDEAL I)
  {
    "ideal generated by";
    I.I;
  }
  proc H_IDEAL_pretty_print(HOMOGENEOUS_IDEAL I)
  {
    "homogeneous ideal generated by";
    I.I;
    "with weights";
    I.weights;
  }
  proc p_print(IDEAL I) { I.prettyprint(I); }
  ring r;
  IDEAL I;
  I.I=ideal(x+y2,z);
  I.prettyprint=IDEAL_pretty_print;
  HOMOGENEOUS_IDEAL H;
  H.I=ideal(x,y,z);
  H.prettyprint=H_IDEAL_pretty_print;
  H.weights=intvec(1,1,1);
  p_print(I);
↦ ideal generated by
↦ _[1]=y2+x
↦ _[2]=z
  p_print(H);
↦ homogeneous ideal generated by
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦ with weights
↦ 1,1,1
```

4.24.2 Declaration of objects of a user defined type

Example:

```
  newstruct("nt","int a,poly b,string c");
  nt A;
  // as long as there is no value assigned to A.b, no ring is needed
  nt B=A;
```

4.24.3 Access to elements of a user defined type

Access to elements of a user defined type via `.`: <object>.<element_name>. The <element_names> are from the definition of the type. Additional, all (potentially) ring dependent elements have an additional entry `r_`<element_name> for the corresponding ring.

Example:

```
  newstruct("nt","int a,poly b,string c");
  nt A;
  3+A.a;
↦ 3
  A.c="example string";
  ring r;
  A.b=poly(1); // assignment: expression must be of the given type
  A;
```

```
↦ c=example string
↦ b=1
↦ a=0
  A.r_b;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

4.24.4 Commands for user defined types

User defined types are normal data types (which do not belong to a ring, even if they have ring dependent parts), so they can be passed as argument to procedures, and received as result from procedures.

In order to apply kernel commands to these types (like `string`, `+`), provide a usual procedure (say `proc p..`) for that task and install it via `system("install",` user_type , kernel_command `,p,` number_of_args `);`. The user_type and kernel_command have to be given as strings. For kernel_command having a variable number of arguments (internal `CMD_M`) use 4 independent of the number of really supplied arguments.

List of available kernel commands and the required number_of_args, some accept several variants and appear therefore at several places:

inplace binary operands: `+,-,*,/,div,%,&,|, [`, number_of_args:2

unary functions: `attrib, bareiss, betti, char, char_series, charstr, cleardenom, close, convhull, defined, deg, degree, denominator, det, dim, dump, ERROR, envelope, execute, facstd, factorize, finduni, gen, getdump, hilb, impart, indepSet, interred, jacob, janet, kbase, killattrib, lead, leadcoef, leadexp, leadmonom, load, ludecomp, maxideal, memory, minbase, minres, monitor, monomial, mult, mstd, nameof, ncols, npars, nrows, numerator, nvars, open, opposite, ord, ordstr, par, pardeg, parstr, preimage, prime, primefactors, prune, qhweight, rank, read, regularity, repart, ringlist, rvar, sba, size, slimgb, sortvec, sqrfree, syz, trace, transpose, twostd, typeof, univariate, var, variables, varstr, vdim, waitfirst, waitall, weight`

functions with 2 arguments: `attrib, betti, bracket, chinrem, coeffs, contract, deg, delete, diff, dim, extgcd, eliminate, exportto, facstd, factorize, farey, fetch, fglm, fglmquot, find, fres, frwalk, gcd, hilb, homog, hres, imap, importfrom, indepSet, insert, interpolation, janet, kbase, kernel, killattrib, koszul, lift, liftstd, load, lres, modulo, mpresmat, mres, newstruct, nc_algebra, nres, oppose, parstr, primefactors, quotient, random, rank, read, sba, simplify, sqrfree, sres, varstr, waitfirst, waitall, wedge`

functions with 3 arguments: `attrib, bareiss, coeffs, eliminate, find, fres, frwalk, hilb, homog, insert, koszul, laguerre, lift, liftstd, newstruct, preimage, random, resultant, sba, vandermonde`

functions with variable number of arguments arguments (number_of_args:4): `breakpoint, coef, dbprint, division, factmodd, intersect, jet, luinverse, lusolve, minor, names, option, qrds, reduce, reservedName, simplex, status, std, subst, system, test, uressolve, write`

Example:

```
  newstruct("nt","int a,poly b,string c");
  nt A;
  A;
↦ c=
↦ b=??
↦ a=0
  ring r;
  // a pretty print routine for nt:
  proc pretty_print(nt A)
  {
    "nt with string c:"+A.c+" and poly:"+string(A.b);
  }
  system("install","nt","print",pretty_print,1); // default printing uses print
  A;
↦ nt with string c: and poly:0
↦ 
  // a custem add for nt:
  proc nt_add(nt A,nt B)
  {
    nt C;
    C.a=A.a+B.a; C.b=A.b+B.b; C.c=A.c+B.c;
    return(C);
  }
  system("install","nt","+",nt_add,2);
  A.b=x;
  nt B; B.c="B"; B.b=y;
  A+B;
↦ nt with string c:B and poly:x+y
↦ 
```

4.24.5 Assignments for user defined types

By default, only objects of the same (user defined) type can be assigned, there is no automatic type conversion as for the kernel data types.

But the operator `=` can be overridden in oder to write custom constructors (the custom constructor does not apply to assignments of the same type): via `system("install",` user_type `,"=" ,p,1);`. The user_type has to be given as a string.

Example:

```
  newstruct("wrapping","poly p");
  proc wrap(poly p)
  {
    wrapping w; w.p = p;
    return  (w);
  }
  system("install", "wrapping", "=", wrap, 1);
  ring r = 0,x,dp;
  wrapping w = x+1;
  w;
↦ p=x+1
  w = int(1); // via conversion int->poly
  w;
```

```
↦ p=1
  w=number(2); // via conversion number->poly
  w;
↦ p=2
```

The user defined procedure for `=` provides also generic type conversions: `hh A=hh(b);` is equivalent to `hh tmp=b; hh A=tmp; kill tmp;`.

4.25 cone

In order to use convex objects in Singular, Singular has to be build from sources together with gfanlib, a C++ library for convex geometry by Anders N. Jensen. Please check the readme file for installation instructions.

This version of Singular does not support `cone`.

4.26 fan

Not supported in this version of Singular

4.27 polytope

Not supported in this version of Singular

Not supported in this version of Singular

4.28 pyobject

The pyobject is a black box data type in Singular for handling objects from the programming language `python`. It needs the `python` support of Singular to be installed.

Together with some basic operations and functions, pyobject instances access `python` functionality from within Singular and store the results for re-use:

Note that this feature is automatically loaded on demand when initializing an object of type `pyobject`. For accessing `pyobject`-related functions before using any `python` object, please type `LIB("pyobject.so");` at the Singular prompt.

```
pyobject pystr = "Hello";
pyobject pyint = 2;
string singstr = string(pystr + " World!");
singstr;
↦ 'Hello World!'
pystr + pyint;  // Error: not possible
↦    ? pyobject error occurred
↦    ? cannot concatenate 'str' and 'int' objects
↦    ? error occurred in or before ./examples/pyobject.sing line 5: `pystr \
   + pyint;  // Error: not possible`
pystr * pyint;  // But this is allowed,
↦ 'HelloHello'
pystr * 3;      // as well as this;
↦ 'HelloHelloHello'

python_run("def newfunc(*args): return list(args)");  // syncs contexts!
newfunc(1, 2, 3);          // newfunc also knowd to SINGULAR
↦ [1, 2, 3]
```

```
def pylst = python_eval("[3, 7, 1]");
proc(attrib(pylst, "sort"))(); // Access python member routines as attributes
pylst.sort();              // <- equivalent short-notation
pylst."sort"();            // <- alternative short-notation
pylst;
↦ [1, 3, 7]

python_import("os");        // Gets stuff from python module 'os'
name;                       // The identifier of the operating system
↦ 'posix'
```

4.28.1 pyobject declarations

Syntax: pyobject name = pyobject_expression ;

Purpose: defines a python object.

Default: None

Example:

```
  pyobject empty;
  empty;
↦ None

  pyobject pystr = "Hello World!";
  pyobject pyone = 17;
  pyobject pylst = list(pystr, pyone);
  pylst;
↦ ['Hello World!', 17]
```

4.28.2 pyobject expressions

A pyobject expression is (optional parts in square brackets):

1. an identifier of type pyobject
2. a function returning pyobject
3. pyobject expressions combined by the arithmetic operations +, -, *, /, or ^, and the member-of operators . and ::
4. an list expression with elements made of pyobject expressions (see Section 3.5.5 [Type conversion and casting], page 46)
5. an string expression (see Section 3.5.5 [Type conversion and casting], page 46)
6. an int expression (see Section 3.5.5 [Type conversion and casting], page 46)

Example:

```
  pyobject pystr = "python string ";
  pystr;
↦ 'python string '
  pyobject pyint = 2;
  pyint;
↦ 2
  pyobject pylst = list(pystr, pyint);
  pylst;
```

```
↦ ['python string ', 2]
  pyint + pyint;
↦ 4
  pyint * pyint;
↦ 4
  pystr + pystr;
↦ 'python string python string '
  pystr * pyint;
↦ 'python string python string '
  python_eval("17 + 4");
↦ 21
  typeof(_);
↦ pyobject
```

4.28.3 pyobject operations

`+`
: addition

`-`
: negation or subtraction

`*`
: multiplication

`/`
: division

`^`, `**`
: power by a positive integer

`<`, `<=`, `>`, `>=`, `==`, `<>`
: comparators (considering leading monomials w.r.t. monomial ordering)

pyobject_expression `[` int_expression `]`
: get the item from the pyobject by index

pyobject_expression `(` pyobject_expression_sequence `)`
: call the pyobject with a sequence of python arguments (the latter may be empty)

pyobject_expression `.` (string_expression `|` name),
: pyobject_expression `::` (string_expression `|` name) get attribute (class member) of a python object

Example:

```
pyobject two = 2;
pyobject three = 3;

two + three;
↦ 5
two - three;
↦ -1
two * three;
↦ 6
two / three;
↦ 0
two ^ three;
↦ 8
two ** three;
↦ 8
```

```
three < two;
↦ 0
two < three;
↦ 1
three <= two;
↦ 0
two <= three;
↦ 1
two == three;
↦ 0
two == two;
↦ 1
three > two;
↦ 1
two > three;
↦ 0
three >= two;
↦ 1
two >= three;
↦ 0
two != two;
↦ 0
two != three;
↦ 1

pyobject pystr = "Hello";
pystr + " World!";
↦ ’Hello World!’
pystr * 3;
↦ ’HelloHelloHello’
pystr[1];
↦ ’e’

python_run("def newfunc(*args): return list(args)");
newfunc();
↦ []
newfunc(two, three);
↦ [2, 3]

newfunc."__class__";
↦ <type ’function’>
newfunc::"__class__";
↦ <type ’function’>
newfunc.func_name;
↦ ’newfunc’
newfunc::func_name;
↦ ’newfunc’
```

4.28.4 pyobject related functions

`attrib` list, get and set attributes (class members) of a pyobject (see Section 5.1.2 [attrib], page 156)

Example:

```
    pyobject pystr = "Kublai Khan";

    // Additional functionality through attrib
    attrib(pystr, "__doc__");
  ↦ "str(object='') -> string\n\nReturn a nice string representation of the
     bject.\nIf the argument is a string, the return value is the same objec
     "
    proc(attrib(pystr, "count"))("K");
  ↦ 2

    pystr."__doc__";           // <- Short notations
  ↦ "str(object='') -> string\n\nReturn a nice string representation of the
     bject.\nIf the argument is a string, the return value is the same objec
     "
    pystr.count("a");          // Even shorter (if attribute's name is valid
  ↦ 2

    python_run("def func(): return 17");
    attrib(func);
  ↦ ['__call__', '__class__', '__closure__', '__code__', '__defaults__', '_
     elattr__', '__dict__', '__doc__', '__format__', '__get__', '__getattribu
     e__', '__globals__', '__hash__', '__init__', '__module__', '__name__',
     _new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__si
     eof__', '__str__', '__subclasshook__', 'func_closure', 'func_code', 'fur
     _defaults', 'func_dict', 'func_doc', 'func_globals', 'func_name']
    attrib(func, "func_name");
  ↦ 'func'
    attrib(func, "func_name", "byAnyOtherName");
    attrib(func, "func_name");
  ↦ 'byAnyOtherName'
```

`killattrib`

deletes an attribute from a pyobject (see Section 5.1.72 [killattrib], page 207)

Example:

```
  LIB("pyobject.so");
  python_run("def new_pyobj(): pass");
  attrib(new_pyobj, "new_attr", "something");
  attrib(new_pyobj, "new_attr");
  ↦ 'something'
  attrib(new_pyobj);
  ↦ ['__call__', '__class__', '__closure__', '__code__', '__defaults__', '_
     elattr__', '__dict__', '__doc__', '__format__', '__get__', '__getattribu
     e__', '__globals__', '__hash__', '__init__', '__module__', '__name__',
     _new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__si
     eof__', '__str__', '__subclasshook__', 'func_closure', 'func_code', 'fur
     _defaults', 'func_dict', 'func_doc', 'func_globals', 'func_name', 'new_a
     tr']

  killattrib(new_pyobj, "new_attr");
  attrib(new_pyobj);
```

```
↦ ['__call__', '__class__', '__closure__', '__code__', '__defaults__', '_
  elattr__', '__dict__', '__doc__', '__format__', '__get__', '__getattribu
  e__', '__globals__', '__hash__', '__init__', '__module__', '__name__', 
  _new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__s
  eof__', '__str__', '__subclasshook__', 'func_closure', 'func_code', 'fu
  _defaults', 'func_dict', 'func_doc', 'func_globals', 'func_name']
```

`python_run`
: execute string-given **python** commands and import new symbols from **python** to SINGULAR's context (see Section 4.28.7 [python_run], page 146).

`python_eval`
: evaluate a string-given **python** expression and return the result to SINGULAR (see Section 4.28.5 [python_eval], page 145).

`python_import`
: import **python** module into SINGULAR's context (see Section 4.28.6 [python_import], page 145)

4.28.5 python_eval

Syntax: `python_eval` (string_expression)

Type: pyobject

Purpose: Evaluates a python expression (given as a string) and returns the result as pyobject.

Example:

```
  LIB("pyobject.so");
  python_eval("17 + 4");
↦ 21
  typeof(_);
↦ pyobject
  list ll = python_eval("range(10)");
```

4.28.6 python_import

Syntax: `python_import` (string_expression)

Type: pyobject

Purpose: Imports python module (given as a string) in the SINGULAR context.

Example:

```
  LIB("pyobject.so");
  python_import("os");
  name;                          // e. g. 'posix'
↦ 'posix'
  sep;                           // pathname separator
↦ '/'
  linesep;                       // end of line marker
↦ '\n'
```

4.28.7 python_run

Syntax: `python_run` (string_expression)

Type: none

Purpose: Executes python commands (given as a string) in `python` context and syncs the contexts afterwards.

Example:

```
  LIB("pyobject.so");
  python_run("def newfunc(*args): return list(args)");
  newfunc(1, 2, 3);          // newfunc also known to SINGULAR now
↦ [1, 2, 3]

  python_run("import os");
  os;
↦ <module 'os' from '/usr/lib64/python2.7/os.pyc'>
  attrib(os, "name");
↦ 'posix'
```

4.29 reference and shared (experimental)

The black box data types `reference` and `shared` in Singular allow for concurrently accessing Singular object data. Copying such object will only add an additional handle which allows you to define multiple identifiers for the same object instance.

Both experimental features are hidden by default, please activate them by typing `system("reference");` or `system("shared");`, respectively, at the Singular prompt.

You must initialize a `reference` using a named identifier or a subexpression of the latter. The resulting object can be stored to gain read and write access from sophisticated data structures.

```
system("reference"); system("shared");
int i = 17;
reference ref = i;

ref;
↦ 17
↦
ref = 19;
ref;
↦ 19
↦
i;              // original handle changed!
↦ 19

kill ref;
i;              // 'i' stays alive
↦ 19

reference uninitialized;
uninitialized;   // not initialized
↦ <unassigned reference or shared memory>
// error: not a named identifier:
uninitialized = 17;
```

```
↦    ? Can only take reference from identifier
↦    ? error occurred in or before ./examples/reference_and_shared__experim\
   ental_.sing line 16: ‘uninitialized = 17;‘

// but subexpressions of named identifiers will do
list ll = list(3,4,5);
reference ref = ll[2];
ref;
↦ 4
↦
ref = 12;
ref;
↦ 12
↦
ll;
↦ [1]:
↦    3
↦ [2]:
↦    12
↦ [3]:
↦    5
```

In contrast, the type `shared` can be used to avoid the initial identifier definition. Each copy has equal rights for manipulating the data.

```
system("reference"); system("shared");
shared ll= list(2,3);

ll[1];
↦ 2
↦
ll[1]= 17;
ll;
↦ [1]:
↦    17
↦ [2]:
↦    3
↦
```

In most cases the value look-up is done automatically, but sometimes you have to disambiguate the input.

```
system("reference"); system("shared");
int i = 0;
reference ref = i;
shared sh = 12;

ref + sh;    // automated 'dereferencing'
↦ 12
ref + 4;
↦ 4
4 + sh;
↦ 16

list ll = list(ref, ref, ref, ref, ref, ref, ref);
string(ll);
```

```
↦ 0,0,0,0,0,0,0
ref = 1;
string(ll);   // all one now
↦ 1,1,1,1,1,1,1

ll[3] = 0;
string(ll);   // only third element changed
↦ 1,1,0,1,1,1,1

reference(ll[1]) = 9;
string(ll);   // all others changed
↦ 9,9,0,9,9,9,9

def(ll[1]) = 11;  // alternative (generic) syntax
string(ll);
↦ 11,11,0,11,11,11,11
```

The previous example had shown that `reference` and `shared` objects can store highly structured without duplicating data all over again. As an additional feature, you can use `reference` objects for implementing procedures having side-effects.

```
system("reference"); system("shared");
list changeme;
changeme;
↦ empty list

proc setfirst(reference ll, def arg) {  ll[1] = arg; }

setfirst(changeme, 17);
changeme;
↦ [1]:
↦    17
```

If you do not need write-access to `proc` parameters, your code will usually perform better using the `alias` statement in the parameter list, see Section 4.18 [proc], page 124.

4.29.1 reference declarations

Syntax: reference name = identifier ;

Purpose: defines a `reference` object.

Default: `None`

Example:

```
system("reference"); system("shared");
  reference empty;
  empty;
↦ <unassigned reference or shared memory>

  string str = "Hello World!";
  reference ref = str;
  ref;
↦ Hello World!
↦
  ref = 17;    // cannot change type of 'i'
```

```
↦    ? ‘string‘(str) = ‘int‘ is not supported
↦    ? expected ‘string‘ = ‘string‘
↦    ? error occurred in or before ./examples/reference_declarations.sing
   ine 8: ‘  ref = 17;    // cannot change type of ’i’‘
  list ll= list(4, 5, 6);
  reference lref = ll[2];
  lref;
↦ 5
↦
  lref = str;  // change list element
  ll;
↦ [1]:
↦    4
↦ [2]:
↦    Hello World!
↦ [3]:
↦    6
```

4.29.2 reference expressions

A reference expression:

1. any identifier
2. any subexpression of an identifier
3. an object of type `reference` (result will reference the original identifier, too)

Example:

```
system("reference"); system("shared");
  int i = 17;
  reference ref = i;  // new reference
  ref;
↦ 17
↦
  reference second = ref;
  second;
↦ 17
↦
  second = 9;       // also tied to ’i’
  i;
↦ 9
  typeof(ref);
↦ reference

  list ll = list(1, 2, 3);
  reference lref = ll[1];
  lref;
↦ 1
↦
  lref = 12;
  ll;
↦ [1]:
↦    12
↦ [2]:
```

```
↦    2
↦ [3]:
↦    3
```

4.29.3 shared declarations

Syntax: shared name = expression ;

Purpose: defines a `shared` object.

Default: `None`

Example:

```
system("reference"); system("shared");
  shared empty;
  empty;
↦ ‘_‘
↦

  shared str = "Hello World!";
  str;
↦ Hello World!
↦
  shared ll= list(4, 5, 6);
  ll;
↦ [1]:
↦    4
↦ [2]:
↦    5
↦ [3]:
↦    6
↦
  ll[2] = str;  // change list element
  ll;
↦ [1]:
↦    4
↦ [2]:
↦    Hello World!
↦ [3]:
↦    6
↦
```

4.29.4 shared expressions

shared expression:

1. any expression
2. an object of type `shared` (result will reference the same data)

Example:

```
system("reference"); system("shared");
  shared sh = 17;  // new shared
  shared second = sh;
  second;
```

```
↦ 17
↦
  second = 9;       // also tied to 'sh'
  sh;
↦ 9
↦
  typeof(sh);
↦ shared

  shared ll = list(1, 2, 3);
  shared lref = ll[1];
  lref;
↦ 1
↦
  lref = 12;
  ll;
↦ [1]:
↦    12
↦ [2]:
↦    2
↦ [3]:
↦    3
↦
```

4.29.5 reference and shared operations

All operations of the underlying objects are forwarded by `reference` and `shared` objects. THis kind of dereferencing is done automatically in most cases:

Example:

```
system("reference"); system("shared");
int i = 2;
reference two = i;
shared three = 3;

two * three;
↦ 6
two ^ three;
↦ 8
two ** three;
↦ 8

two + two;
↦ 4
two - two;
↦ 0

ring r = 0, (x,y,z), dp;
poly p = x + y + z;
reference ref = p;
shared zvar =z;
subst(ref, x,1, y,2, zvar,3);
↦ 6
```

In some cases references have to be dereferenced explicitly. For instance, this is the case for n-ary function calls not starting with a reference or shared object. You can use the link operator or a type cast to work around this. In contrast, some constructs like left-hand subexpressions prematurely evaluate. You can avoid this by using the def operator or by explicitly type casting to reference.

```
system("reference"); system("shared");
ring r = 0, (x,y,z), dp;
poly p = x + y + z;
shared xsh = x;
subst(p, xsh,1, y,2, z,3);        // fails
↦    ? subst(‘poly‘,‘shared‘,‘int‘) failed
↦    ? expected subst(‘poly‘,‘poly‘,‘poly‘)
↦    ? expected subst(‘matrix‘,‘poly‘,‘int‘)
↦    ? error occurred in or before ./examples/reference_and_shared_operatio\
   ns_1.sing line 5: ‘subst(p, xsh,1, y,2, z,3);        // fails‘
subst(p, poly(xsh),1, y,2, z,3);  // good
↦ 6
subst(p, link(xsh),1, y,2, z,3);  // fine
↦ 6

list ll = list(xsh, xsh, xsh);
ll[1] = y;      // replaced only first entry
ll;
↦ [1]:
↦    y
↦ [2]:
↦    x
↦
↦ [3]:
↦    x
↦
shared(ll[2]) = z;    // replaces the others
ll;
↦ [1]:
↦    y
↦ [2]:
↦    z
↦
↦ [3]:
↦    z
↦
def(ll[2]) = x;       // generic alternative
ll;
↦ [1]:
↦    y
↦ [2]:
↦    x
↦
↦ [3]:
↦    x
↦
```

In particular, explicit dereferencing is useful to distinguish between typecasting and nested constructings.

```
system("reference"); system("shared");
shared shl = list(1);
shl;
↦ [1]:
↦    1
↦
list(shl);  // wraps 'shl' by a list
↦ [1]:
↦    [1]:
↦    1
↦
link(shl);  // extract the list in 'shl'
↦ [1]:
↦    1
```

4.29.6 reference and shared related functions

`def` explicitly type casts to `reference` or `shared`, respectively. (Note: For the `def` declaration, see Section 4.5 [def], page 79.)

Example:

```
system("reference"); system("shared");
int i =1;
reference ref = i;
shared sh = 17;
list ll = list(ref, ref, ref, sh, sh);
ll[1] = 2;      // replace only one entry
ll;
↦ [1]:
↦    2
↦ [2]:
↦    1
↦
↦ [3]:
↦    1
↦
↦ [4]:
↦    17
↦
↦ [5]:
↦    17
↦
def(ll[2]) = 3;      // change the others
ll;
↦ [1]:
↦    2
↦ [2]:
↦    3
↦
↦ [3]:
```

```
          ↦    3
          ↦
          ↦ [4]:
          ↦    17
          ↦
          ↦ [5]:
          ↦    17
          ↦
          def(ll[4]) = 19;     // same here
          ll;
          ↦ [1]:
          ↦    2
          ↦ [2]:
          ↦    3
          ↦
          ↦ [3]:
          ↦    3
          ↦
          ↦ [4]:
          ↦    19
          ↦
          ↦ [5]:
          ↦    19
          ↦
```

`link` explicitly dereference a `reference` or `shared` object. (Note: For the `link` declaration, see Section 4.10 [link], page 96.)

Example:

```
system("reference"); system("shared");
ring r = 0, (x,y,z), dp;
poly p = x + y + z;
def x_=x;
reference xref=x_;
xref;
↦ x
↦
subst(p, xref,1, y,2, z,3);        // fails
↦    ? subst(`poly`,`reference`,`int`) failed
↦    ? expected subst(`poly`,`poly`,`poly`)
↦    ? expected subst(`matrix`,`poly`,`int`)
↦    ? error occurred in or before ./examples/reference_and_shared_relate
   functions_1.sing line 7: `subst(p, xref,1, y,2, z,3);        // fails`
subst(p, link(xref),1, y,2, z,3);  // fine
↦ 6
```

`system` The `reference` and `shared` objects overload the `system` command to gain extended features, see `system(ref, "help")` for more details. (Note: For the general `system` command, see Section 5.1.155 [system], page 275.)

Example:

```
system("reference"); system("shared");
shared sh;
```

```
system(sh, "help");
↦ system(<ref>, ...): extended functionality for reference/shared data <re
  >
↦   system(<ref>, count)         - number of references pointing to <ref>
↦   system(<ref>, enumerate)     - unique number for identifying <ref>
↦   system(<ref>, undefined)     - checks whether <ref> had been assigned
↦   system(<ref>, "help")        - prints this information message
↦   system(<ref>, "typeof")      - actual type referenced by <ref>
↦   system(<ref1>, same, <ref2>) - tests for identic reference objects
```

5 Functions and system variables

5.1 Functions

This section gives a complete reference of all functions, commands and special variables of the Singular kernel (i.e., all built-in commands). See Section D.1 [standard_lib], page 793, for those functions from the `standard.lib` (this library is automatically loaded at start-up time) which extend the functionality of the kernel and are written in the Singular programming language.

The general syntax of a function is

[target =] function_name (<arguments>);

If no target is specified, the result is printed. In some cases (e.g., `export`, `keepring`, `setring`, `type`) the brackets are optional. For the commands `kill`, `help`, `break`, `quit`, `exit` and `LIB` no brackets are allowed.

5.1.1 align

Syntax: `align` (vector_expression, int_expression)
`align` (module_expression, int_expression)

Type: type of the first argument

Purpose: maps module generators `gen(i)` to `gen(i+s)` for all i.

Example:

```
  ring r=0,(x,y,z),(c,dp);
  align([1,2,3],3);
↦ [0,0,0,1,2,3]
  align([0,0,1,2,3],-1);
↦ [0,1,2,3]
  align(freemodule(2),1);
↦ _[1]=[0,1]
↦ _[2]=[0,0,1]
```

5.1.2 attrib

Syntax: `attrib` (name)

Type: none

Purpose: displays the attribute list of the object called name.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=std(maxideal(2));
  attrib(I);
↦ attr:isSB, type int
```

Syntax: `attrib` (name , string_expression)

Type: any

Purpose: returns the value of the attribute string_expression of the variable name. If the attribute is not defined for this variable, `attrib` returns the empty string.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=std(maxideal(2));
  attrib(I,"isSB");
↦ 1
  // maxideal(2) is a standard basis,
  // SINGULAR does know it for maxideal:
  attrib(maxideal(2), "isSB");
↦ 1
```

Syntax: `attrib ( name, string_expression, expression )`

Type: none

Purpose: sets the attribute string_expression of the variable name to the value expression.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=maxideal(2); // the attribute "isSB" is not set
  vdim(I);
↦ 4
  attrib(I,"isSB",0);  // the standard basis attribute is reset here
  vdim(I);
↦ // ** I is no standard basis
↦ 4
```

Remark: An attribute may be described by any string_expression. Some of these are used by the kernel of SINGULAR and referred to as reserved attributes. Non-reserved attributes may be used, however, in procedures and can considerably speed up computations.

Reserved attributes:

(cf_class, global, isSB, isHomog, rank, ring_cf, rowShift are used by the kernel, the other are used by libraries)

`cf_class (for ring)`
: the internal type of the coefficients (see `n_coeffType`)

`global (for ring)`
: 1, if the ordering is global

`isSB (for ideal, module)`
: the standard basis property is set by all commands computing a standard basis like `groebner`, `std`, `stdhilb` etc.; used by `lift`, `dim`, `degree`, `mult`, `hilb`, `vdim`, `kbase`

`isHomog (for ideal, module)`
: the weight vector of module generators for homogeneous or quasihomogeneous ideals/modules,
: used by `betti`, `degree`, `highcorner`, `hilbert`, `homog`, `prune`, `prune_map`, `sba`, `slimgb`, `std`, `syz`, `kbase`, `modulo`, `mres`, `mres_map`, `nres`, `stdhilb`.

`isCI` complete intersection property

`isCM` Cohen-Macaulay property

`maxExp (for ring/list from ringlist)`
: limit for each exponent (32767 by default)

`rank (for module)`
: set/get the rank of a module (see Section 5.1.107 [nrows], page 232)

`ring_cf (for ring)`
the coefficients of the polynomial ring are considered to be a ring

`withSB` value of type ideal, resp. module, is std

`withHilb` value of type intvec is hilb(_,1) (see Section 5.1.56 [hilb], page 195)

`withRes` value of type list is a free resolution

`withDim` value of type int is the dimension (see Section 5.1.25 [dim], page 174)

`withMult` value of type int is the multiplicity (see Section 5.1.101 [mult], page 228)

See Section 5.1.72 [killattrib], page 207.

5.1.3 bareiss

qcindex Gauss

Syntax: `bareiss (` module_expression `)`
`bareiss (` matrix_expression `)`
`bareiss (` module_expression`,` int_expression`,` int_expression `)`
`bareiss (` matrix_expression`,` int_expression`,` int_expression `)`

Type: list of module and intvec

Purpose: applies the sparse Gauss-Bareiss algorithm (see Section C.9 [References], page 791, Lee and Saunders) to a module (or with type conversion to a matrix) with an 'optimal' pivot strategy. The vectors of the module are the columns of the matrix, hence elimination takes place w.r.t. rows.
With only one parameter a complete elimination is done. Result is a list: the first entry is a module with a minimal independent set of vectors (as a matrix lower triangular), the second entry an intvec with the permutation of the rows w.r.t. the original matrix, that is, a k at position l indicates that row k was carried over to the row l.
The further parameters control the algorithm. `bareiss(M,i,j)` does not attempt to diagonalize the last i rows in the elimination procedure and stops computing when the remaining number of vectors (columns) to reduce is at most j.

Example:

```
  ring r=0,(x,y,z),(c,dp);
  module mm;
  // ** generation of the module mm **
  int d=7;
  int b=2;
  int db=d-b;
  int i;
  for(i=d;i>0;i--){ mm[i]=3*x*gen(i); }
  for(i=db;i;i--){ mm[i]=mm[i]+7*y*gen(i+b); }
  for(i=d;i>db;i--){ mm[i]=mm[i]+7*y*gen(i-db); }
  for(i=d;i>b;i--){ mm[i]=mm[i]+11*z*gen(i-b); }
  for(i=b;i;i--){ mm[i]=mm[i]+11*z*gen(i+db); }
  // ** the generating matrix of mm **
  print(mm);
↦ 3x, 0,  11z,0,  0,  7y, 0,
↦ 0,  3x, 0,  11z,0,  0,  7y,
↦ 7y, 0,  3x, 0,  11z,0,  0,
↦ 0,  7y, 0,  3x, 0,  11z,0,
↦ 0,  0,  7y, 0,  3x, 0,  11z,
↦ 11z,0,  0,  7y, 0,  3x, 0,
↦ 0,  11z,0,  0,  7y, 0,  3x
  // complete elimination
  list ss=bareiss(mm);
  print(ss[1]);
↦ 7y, 0,     0,     0,       0,         0,     0,
↦ 3x, -33xz, 0,     0,       0,         0,     0,
↦ 11z,-121z2,1331z3,0,       0,         0,     0,
↦ 0,  0,     0,     9317yz3,0,         0,     0,
↦ 0,  21xy,  _[5,3],14641z4,-43923xz4,0,     0,
↦ 0,  0,     0,     0,       65219y2z3,_[6,6],0,
↦ 0,  49y2,  _[7,3],3993xz3,_[7,5],   _[7,6],_[7,7]
  ss[2];
↦ 2,7,5,1,4,3,6
  // elimination up to 3 vectors
  ss=bareiss(mm,0,3);
  print(ss[1]);
↦ 7y, 0,     0,     0,       0,         0,         0,
↦ 3x, -33xz, 0,     0,       0,         0,         0,
↦ 11z,-121z2,1331z3,0,       0,         0,         0,
↦ 0,  0,     0,     9317yz3,0,         0,         0,
↦ 0,  0,     0,     0,       27951xyz3,102487yz4,65219y2z3,
↦ 0,  21xy,  _[6,3],14641z4,_[6,5],   _[6,6],   -43923xz4,
↦ 0,  49y2,  _[7,3],3993xz3,_[7,5],   _[7,6],   _[7,7]
  ss[2];
↦ 2,7,5,1,3,4,6
  // elimination without the last 3 rows
  ss=bareiss(mm,3,0);
  print(ss[1]);
↦ 7y, 0,   0,      0,       0,     0,     0,
↦ 0,  77yz,0,      0,       0,     0,     0,
↦ 0,  0,   231xyz, 0,       0,     0,     0,
↦ 0,  0,   0,      1617xy2z,0,     0,     0,
↦ 11z,21xy,-1331z3,14641z4, _[5,5],_[5,6],_[5,7],
↦ 0,  0,   539y2z, _[6,4],  _[6,5],_[6,6],-3773y3z,
↦ 3x, 49y2,-363xz2,3993xz3, _[7,5],_[7,6],_[7,7]
  ss[2];
↦ 2,3,4,1
```

See Section 5.1.23 [det], page 172; Section 4.13 [matrix], page 108.

5.1.4 betti

Syntax: `betti` (list_expression)
`betti` (resolution_expression)
`betti` (list_expression , int_expression)
`betti` (resolution_expression , int_expression)

Type: intmat

Purpose: with 1 argument: computes the graded Betti numbers of a minimal resolution of R^n/M, if R denotes the basering, M is a homogeneous submodule of R^n and the argument represents a resolution of R^n/M.

The entry d of the intmat at place (i,j) is the minimal number of generators in degree i+j of the j-th syzygy module (= module of relations) of R^n/M, i.e. the 0th (resp. 1st) syzygy module of R^n/M is R^n (resp. M). The argument is considered to be the result of a res/fres/sres/mres/mres_map/nres/lres command. This implies that a zero is only allowed (and counted) as a generator in the first module.
For the computation betti uses only the initial monomials. This could lead to confusing results for a non-homogeneous input.

If the optional second argument is non-zero, the Betti numbers will be minimized.
`betti` sets the attribute `rowShift`.

Example:

```
  ring r=32003,(a,b,c,d),dp;
  ideal j=bc-ad,b3-a2c,c3-bd2,ac2-b2d;
  list T=mres(j,0); // 0 forces a full resolution
  // a minimal set of generators for j:
  print(T[1]);
↦ bc-ad,
↦ c3-bd2,
↦ ac2-b2d,
↦ b3-a2c
  // second syzygy module of r/j which is the first
  // syzygy module of j (minimal generating set):
  print(T[2]);
↦ bd,c2,ac,b2,
↦ -a,-b,0, 0,
↦ c, d, -b,-a,
↦ 0, 0, -d,-c
  // the second syzygy module (minimal generating set):
  print(T[3]);
↦ -b,
↦ a,
↦ -c,
↦ d
  print(T[4]);
↦ 0
  betti(T);
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,3,4,1
  // most useful for reading off the graded Betti numbers:
  print(betti(T),"betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     1     -     -
↦     2:     -     3     4     1
↦ ------------------------------
↦ total:     1     4     4     1
↦
```

Hence,

- the 0th syzygy module of r/j (which is r) has 1 generator in degree 0 (which is 1),

- the 1st syzygy module T[1] (which is j) has 4 generators (one in degree 2 and three in degree 3),
- the 2nd syzygy module T[2] has 4 generators (all in degree 4),
- the 3rd syzygy module T[3] has 1 generator in degree 5,

where the generators are the columns of the displayed matrix and degrees are assigned such that the corresponding maps have degree 0:

$$0 \longleftarrow r/j \longleftarrow r(1) \xleftarrow{T[1]} r(2) \oplus r^3(3) \xleftarrow{T[2]} r^4(4) \xleftarrow{T[3]} r(5) \longleftarrow 0 \quad .$$

See Section C.3 [Syzygies and resolutions], page 775; Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 5.1.83 [lres], page 215; Section 5.1.98 [mres], page 225; Section 5.1.99 [mres_map], page 226; Section 5.1.120 [print], page 242; Section 5.1.134 [res], page 253; Section 4.19 [resolution], page 125; Section 5.1.149 [sres], page 269.

5.1.5 char

Syntax: char (ring_name)

Type: int

Purpose: returns the characteristic of the coefficient field of a ring.

Example:

```
  ring r=32003,(x,y),dp;
  char(r);
↦ 32003
  ring s=0,(x,y),dp;
  char(s);
↦ 0
  ring ra=(7,a),(x,y),dp;
  minpoly=a^3+a+1;
  char(ra);
↦ 7
  ring rp=(49,a),(x,y),dp;
  char(rp);
↦ 7
  ring rr=real,x,dp;
  char(rr);
↦ 0
```

See Section 5.1.7 [charstr], page 162; Section 4.20 [ring], page 127.

5.1.6 char_series

Syntax: char_series (ideal_expression)

Type: matrix

Purpose: the rows of the matrix represent the irreducible characteristic series of the ideal with respect to the current ordering of variables.
One application is the decomposition of the zero-set.

Example:

```
  ring r=32003,(x,y,z),dp;
  print(char_series(ideal(xyz,xz,y)));
↦ y,z,
↦ x,y
```

See Section C.4 [Characteristic sets], page 776.

5.1.7 charstr

Syntax: `charstr` (ring_name)

Type: string

Purpose: returns the description of the coefficient field of a ring. (Tests for certain types of coefficients should use the routines from `ring.lib` as the string representation may change.)

Example:

```
  ring r=32003,(x,y),dp;
  charstr(r);
↦ ZZ/32003
  ring s=0,(x,y),dp;
  charstr(s);
↦ QQ
  ring ra=(7,a),(x,y),dp;
  minpoly=a^3+a+1;
  charstr(ra);
↦ 7,a
  ring rp=(49,a),(x,y),dp;
  charstr(rp);
↦ 49,a
  ring rr=real,x,dp;
  charstr(rr);
↦ Float()
```

See Section 5.1.5 [char], page 161; Section 5.1.113 [ordstr], page 239; Section 4.20 [ring], page 127; Section D.2.12 [ring_lib], page 942; Section 5.1.167 [varstr], page 285.

5.1.8 chinrem

Syntax: `chinrem` (list, intvec)
`chinrem` (list, list)
`chinrem` (intvec, intvec)

Type: the same type as the elements of the first argument
If the elements of the first argument are lists again, chinrem is applied recursively.

Purpose: applies chinese remainder theorem to the first argument w.r.t. the moduli given in the second. The elements in the first list must be of same type which can be `bigint`/`int`, `poly`, `ideal`, `module`, `smatrix` or `matrix`. The moduli, if given by a list, must be of type `bigint` or `int`.
If data depending on a ring are involved, the coefficient field must be `Q`.

Example:

```
  chinrem(intvec(2,-3),intvec(7,11));
↦ 30
  chinrem(list(2,-3),list(7,11));
↦ 30
  ring r=0,(x,y),dp;
  ideal i1=5x+2y,x2+3y2+xy;
  ideal i2=2x-3y,2x2+4y2+5xy;
  chinrem(list(i1,i2),intvec(7,11));
↦ _[1]=-9x+30y
↦ _[2]=-20x2-6xy-18y2
  chinrem(list(i1,i2),list(bigint(7),bigint(11)));
↦ _[1]=-9x+30y
↦ _[2]=-20x2-6xy-18y2
  chinrem(list(list(i1,i2),list(i1,i2)),list(bigint(7),bigint(11)));
↦ [1]:
↦    _[1]=-9x+30y
↦    _[2]=-20x2-6xy-18y2
↦ [2]:
↦    _[1]=-9x+30y
↦    _[2]=-20x2-6xy-18y2
```

See Section D.4.16 [modstd_lib], page 1146.

5.1.9 cleardenom

Syntax: `cleardenom` (poly_expression)
`cleardenom` (vector_expression)

Type: same as the input type

Purpose: multiplies a polynomial, resp. vector, by a suitable constant to cancel all denominators from its coefficients and then divide it by its content.

Example:

```
  ring r=0,(x,y,z),dp;
  poly f=(3x+6y)^5;
  f/5;
↦ 243/5x5+486x4y+1944x3y2+3888x2y3+3888xy4+7776/5y5
  cleardenom(f/5);
↦ x5+10x4y+40x3y2+80x2y3+80xy4+32y5
  vector w= [4x2+20,6x+2,0,8];   // application to a vector
  print(cleardenom(w));
↦ [2x2+10,3x+1,0,4]
```

See Section D.2.8.14 [content], page 894.

5.1.10 close

Syntax: `close` (link_expression)

Type: none

Purpose: closes a link.

Example:

```
link l="ssi:tcp localhost:"+system("Singular");
open(l); // start SINGULAR "server" on localhost in batchmode
close(l); // shut down SINGULAR server
```

See Section 4.10 [link], page 96; Section 5.1.110 [open], page 234.

5.1.11 coef

Syntax: `coef` (poly_expression`,` product_of_ringvars)
`coef` (ideal_expression`,` product_of_ringvars)

Type: matrix

Syntax: `coef` (vector_expression`,` product_of_ringvars`,` matrix_name`,` matrix_name)

Type: none

Purpose: determines the monomials in f divisible by a ring variable of m (where f is the first argument and m the second argument) and the coefficients of these monomials as polynomials in the remaining variables. First case: returns a $2 \times n$ matrix M, n being the number of the determined monomials. The first row consists of these monomials, the second row of the corresponding coefficients of the monomials in f. Thus, $f = M[1,1] \cdot M[2,1] + \ldots + M[1,n] \cdot M[2,n]$.

Second case: apply to all generators of the ideal and combine the results into one matrix.

Third case: the second matrix (i.e., the 4th argument) contains the monomials, the first matrix (i.e., the 3rd argument) the corresponding coefficients of the monomials in the vector.

Note: coef considers only monomials which really occur in f (i.e., which are not 0), while coeffs (see Section 5.1.12 [coeffs], page 165) returns the coefficient 0 at the appropriate place if a monomial is not present.

Example:

```
  ring r=32003,(x,y,z),dp;
  poly f=x5+5x4y+10x2y3+y5;
  matrix m=coef(f,y);
  print(m);
↦ y5,y3,  y,  1,
↦ 1, 10x2,5x4,x5
  f=x20+xyz+xy+x2y+z3;
  print(coef(f,xy));
↦ x20,x2y,xy, 1,
↦ 1,  1,  z+1,z3
  print(coef(maxideal(3),yz));
↦ y3,y2z,yz2,z3,y2,yz,z2,y, z, 1,
↦ 0, 0,  0,  1, 0, 0, 0, 0, 0, 0,
↦ 0, 0,  1,  0, 0, 0, 0, 0, 0, 0,
↦ 0, 1,  0,  0, 0, 0, 0, 0, 0, 0,
↦ 1, 0,  0,  0, 0, 0, 0, 0, 0, 0,
↦ 0, 0,  0,  0, 0, 0, x, 0, 0, 0,
↦ 0, 0,  0,  0, 0, x, 0, 0, 0, 0,
↦ 0, 0,  0,  0, x, 0, 0, 0, 0, 0,
↦ 0, 0,  0,  0, 0, 0, 0, 0, x2,0,
↦ 0, 0,  0,  0, 0, 0, 0, x2,0, 0,
↦ 0, 0,  0,  0, 0, 0, 0, 0, 0, x3
  vector v=[f,zy+77+xy];
  print(v);
↦ [x20+x2y+xyz+z3+xy,xy+yz+77]
  matrix mc; matrix mm;
  coef(v,y,mc,mm);
  print(mc);
↦ x2+xz+x,x20+z3,
↦ x+z,    77
  print(mm);
↦ y,1,
↦ y,1
```

See Section 5.1.12 [coeffs], page 165.

5.1.12 coeffs

Syntax: `coeffs` (poly_expression , ring_variable)
`coeffs` (ideal_expression, ring_variable)
`coeffs` (vector_expression, ring_variable)
`coeffs` (module_expression, ring_variable)
`coeffs` (poly_expression, ring_variable, matrix_name)
`coeffs` (ideal_expression, ring_variable, matrix_name)
`coeffs` (vector_expression, ring_variable, matrix_name)
`coeffs` (module_expression, ring_variable, matrix_name)

Type: matrix

Syntax: `coeffs` (ring_expression)

Type: cring

Purpose: develops each polynomial of the first argument J as a univariate polynomial in the given ring_variable z, and returns the coefficients as a matrix M.

With e denoting the maximal z-degree occurring in the polynomials of J, and d:=e+1, M $= (m_{ij})$ satisfies the following conditions:

(i) If J is a single polynomial f, then M is a $(d \times 1)$-matrix and $m_{i+1,j}, 0 \leq i \leq e$, is the coefficient of z^i in f.

(ii) If J is an ideal with generators $f_1, f_2, \ldots, f_k$ then M is a $(d \times k)$-matrix and $m_{i+1,j}, 0 \leq i \leq e, 1 \leq j \leq k$, is the coefficient of z^i in f_j.

(iii) If J is a k-dimensional vector with entries $f_1, f_2, \ldots, f_k$ then M is a $(dk \times 1)$-matrix and $m_{(j-1)d+i+1,1}, 0 \leq i \leq e, 1 \leq j \leq k$, is the coefficient of z^i in f_j.

(iV) If J is a module generated by s vectors $v_1, v_2, \ldots, v_s$ of dimension k then M is a $(dk \times s)$-matrix and $m_{(j-1)d+i+1,r}, 0 \leq i \leq e, 1 \leq j \leq k, 1 \leq r \leq s$, is the coefficient of z^i in the j-th entry of v_r.

The optional third argument T can be used to return the matrix of powers of z such that matrix(J) = T*M holds in each of the previous four cases.

Note: `coeffs` returns the coefficient 0 at the appropriate matrix entry if a monomial is not present, while `coef` considers only monomials which actually occur in the given expression.

Example:

```
  ring r;
  poly f = (x+y)^3;
  poly g = xyz+z10y4;
  ideal i = f, g;
  matrix M = coeffs(i, y);
  print(M);
↦ x3, 0,
↦ 3x2,xz,
↦ 3x, 0,
↦ 1,  0,
↦ 0,  z10
  vector v = [f, g];
  M = coeffs(v, y);
  print(M);
↦ x3,
↦ 3x2,
↦ 3x,
↦ 1,
↦ 0,
↦ 0,
↦ xz,
↦ 0,
↦ 0,
↦ z10
```

Syntax: `coeffs` (ideal_expression, ideal_expression)
`coeffs` (module_expression, module_expression)
`coeffs` (ideal_expression, ideal_expression, product_of_ringvars)
`coeffs` (module_expression, module_expression, product_of_ringvars)

Type: matrix

Purpose: expresses each polynomial of the first argument M as a sum $\sum_{i=1}^{k} m_i \cdot a_i \cdot x^{e_i}$, where the m_i come from a specified set of monomials, the a_i are from the underlying coefficient ring (or field), and the x^{e_i} are powers of a specified ring variable x.

The second parameter K provides the set of monomials which should be sufficient to generate all entries of M.
Both M and K can be thought of as the matrices obtained by matrix(M) and matrix(K), respectively. (If M and K are given by ideals, then this matrix has just one row.)

The optional parameter product_of_ringvars determines the variable x: It is expected to be either the product of all ring variables (then x is 1, and each polynomial will be expressed as $\sum_{i=1}^{k} m_i \cdot a_i$, or product_of_ringvars is the product of all ring variables except one variable (which then determines x). If product_of_ringvars is omitted then x = 1 as default.

If K contains all monomials that are necessary to express the entries of M, then the returned matrix A satisfies $K \cdot A = M$. Otherwise only a subset of entries of $K \cdot A$ and M will coincide. In this case, the valid entries start at M[1,1] and run from left to right, top to bottom.

Note: Note that in general not all entries of K*A and M will coincide, depending on the set of monomials provided by K.

Example:

```
  ring r=32003,(x,y,z),dp;
  module M = [y3+x2z, xy], [-xy, y2+x2z];
  print(M);
↦ y3+x2z,-xy,
↦ xy,    x2z+y2
  module K = [x2, xy], [y3, xy], [xy, x];
  print(K);
↦ x2,y3,xy,
↦ xy,xy,x
  matrix A = coeffs(M, K, xy);   // leaving z as variable of interest
  print(A);   // attention: only the first row of M is reproduced by K*A
↦ z,0,
↦ 1,0,
↦ 0,-1
```

Syntax: `coeffs` (ring_expression)

Type: cring

Purpose: return the coefficient ring of the argument

Example:

```
  ring R=QQ,x,dp;
  coeffs(R);
↦ QQ
```

See Section 5.1.11 [coef], page 164; Section 5.1.69 [kbase], page 205.

5.1.13 contract

Syntax: `contract` (ideal_expression, ideal_expression)

Type: matrix

Purpose: contracts each of the n elements of the second ideal J by each of the m elements of the first ideal I, producing an $m \times n$ matrix.
Contraction is defined on monomials by:

$$\text{contract}(x^A, x^B) := \begin{cases} x^{(B-A)}, & \text{if } B \geq A \text{ componentwise} \\ 0, & \text{otherwise.} \end{cases}$$

where A and B are the multiexponents of the ring variables represented by x. `contract` is extended bilinearly to all polynomials.

Example:

```
  ring r=0,(a,b,c,d),dp;
  ideal I=a2,a2+bc,abc;
  ideal J=a2-bc,abcd;
  print(contract(I,J));
↦ 1,0,
↦ 0,ad,
↦ 0,d
```

See Section 5.1.24 [diff], page 173.

5.1.14 create_ring

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Usage: create_ring(l1, l2, l3[, l4, "no_minpoly"]);
l1 int or list, l2 list or string, l3 list or string, l4 ideal

Return: ring(list(l1, l2, l3, l4))

Note: l1, l2, l3, l4 are assumed to be the four entries of ring_list(R) where R is the ring to be returned.
Optional arguments: If l4 is not given, it is assumend to be ideal(0). If "no_minpoly" is given, then the minimal polynomial in l1, if present, is set to 0.
Shortcuts: Strings such as "0", "(32003)" or "(0,a,b,c)" can be given as l1. Indexed parameters as in "(0,a(1..3))" are not supported. Strings such as "(x,y,z)" can be given as l2. Indexed variables as in "(x(1..3),y,z)" are not supported. Strings representing orderings such as "dp" or "(lp(3), ds(2))" can be given as l3, except matrix orderings given by
"M([intmat_expression])".

Example:

```
  ring R = (0,a), x, lp;
ring_list(R);
↦ [1]:
↦ 0,a
↦ [2]:
↦    [1]:
↦       x
↦ [3]:
↦    [1]:
↦       [1]:
↦          lp
↦       [2]:
↦          1
↦    [2]:
↦       [1]:
↦          C
↦       [2]:
↦          0
↦ [4]:
↦    _[1]=0
minpoly = a^2+1;
qring Q = ideal(x^3-2);
ring S = create_ring(ring_list(Q)[1], "(x,y,t)", "dp", "no_minpoly");
basering;
↦ // coefficients: QQ[a]/(a2+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y t
↦ //        block   2 : ordering C
```

5.1.15 crossprod

Syntax: `crossprod` (cring_expression, ...)

Type: cring

Purpose: crooss product of several objects of type cring

Example:

```
          crossprod(ZZ/32003,Float());
        ↦ ZZ/32003 x Float()
```

See Section 4.1 [cring], page 73.

5.1.16 datetime

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `datetime ()`

Return: string

Purpose: return the current date and time as a string

Example:

```
  datetime();
↦ Di 28. Nov 10:02:38  2023
```

5.1.17 dbprint

Syntax: `dbprint (` int_expression`,` expression_list `)`

Type: none

Purpose: applies the print command to each expression in the expression_list if int_expression is positive. `dbprint` may also be used in procedures in order to print results subject to certain conditions.

Syntax: `dbprint (` expression `)`

Type: none

Purpose: The print command is applied to the expression if `printlevel>=voice`.

Note: See Section 3.8 [Libraries], page 55, for an example how this is used for displaying comments while procedures are executed.

Example:

```
          int debug=0;
          intvec i=1,2,3;
          dbprint(debug,i);
          debug=1;
          dbprint(debug,i);
        ↦ 1,
        ↦ 2,
        ↦ 3
          voice;
        ↦ 1
          printlevel;
        ↦ 0
          dbprint(i);
```

See Section 3.9 [Debugging tools], page 68; Section 5.1.120 [print], page 242; Section 5.3.6 [printlevel], page 304; Section 5.3.11 [voice], page 308.

5.1.18 defined

Syntax: `defined` (name)

Type: int

Purpose: returns a value `<>`0 (TRUE) if there is a user-defined object with this name, and 0 (FALSE) otherwise.
A non-zero return value is the level where the object is defined (level 1 denotes the top level, level 2 the level of a first procedure, level 3 the level of a procedure called by a first procedure, etc.). For ring variables and other constants, -1 is returned.

Note: A local object `m` may be identified by `if (defined(m)==voice)`.

Example:

```
  ring r=(0,t),(x,y),dp;
  matrix m[5][6]=x,y,1,2,0,x+y;
  defined(mm);
↦ 0
  defined(r) and defined(m);
↦ 1
  defined(m)==voice;   // m is defined in the current level
↦ 1
  defined(x);
↦ -1
  defined(z);
↦ 0
  defined("z");
↦ -1
  defined(t);
↦ -1
  defined(42);
↦ -1
```

See Section 5.1.139 [rvar], page 258; Section 5.3.11 [voice], page 308.

5.1.19 deg

Syntax: `deg` (poly_expression)
`deg` (vector_expression)
`deg` (poly_expression `,` intvec_expression)
`deg` (vector_expression `,` intvec_expression)

Type: int

Purpose: returns the maximal (weighted) degree of the terms of a polynomial or a vector;
deg(0) is -1.
The optional second argument gives the weight vector, otherwise weight 1 is used for lex orderings and block ordering, the default weights of the base ring are used for orderings consisting of one block.

Example:

```
  ring r=0,(x,y,z),lp;
  deg(0);
↦ -1
  deg(x3+y4+xyz3);
```

```
↦ 5
  ring rr=7,(x,y),wp(2,3);
  poly f=x2+y3;
  deg(f);
↦ 9
  ring R=7,(x,y),ws(2,3);
  poly f=x2+y3;
  deg(f);
↦ 9
  vector v=[x2,y];
  deg(v);
↦ 4
```

See Section 5.1.68 [jet], page 204; Section 5.1.112 [ord], page 238; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.20 degree

Syntax: `degree` (ideal_expression)
`degree` (module_expression)

Type: string

Purpose: computes the (Krull) dimension and the multiplicity of the ideal, resp. module, generated by the leading monomials of the input and prints it. This is equal to the dimension and multiplicity of the ideal, resp. module, if the input is a standard basis with respect to a degree ordering.

Example:

```
ring r3=32003,(x,y,z),ds;
int a,b,c,t=11,10,3,1;
poly f=x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^(c-1)*y^(c-1)*z3
  +x^(c-2)*y^c*(y2+t*x)^2;
ideal i=jacob(f);
ideal i0=std(i);
degree(i0);
↦ // dimension (local)   = 0
↦ // multiplicity = 314
```

See Section 5.1.25 [dim], page 174; Section 4.6 [ideal], page 80; Section 5.1.101 [mult], page 228; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.21 delete

Syntax: `delete` (list_expression, int_expression)
`delete` (intvec_expression, int_expression)
`delete` (ideal_expression, int_expression)
`delete` (module_expression, int_expression)
`delete` (list_expression, intvec_expression)
`delete` (ideal_expression, intvec_expression)
`delete` (module_expression, intvec_expression)

Type: type of the first argument

Purpose: deletes the element(s) with the given index/indices from a list/intvec/ideal/module (the input is not changed).

Example:

```
  list l="a","b","c";
  list l1=delete(l,2);l1;
↦ [1]:
↦    a
↦ [2]:
↦    c
  l;
↦ [1]:
↦    a
↦ [2]:
↦    b
↦ [3]:
↦    c
  delete(1..5,2);
↦ 1,3,4,5
  ring r=0,(x,y,z),dp;
  delete(maxideal(1),1);
↦ _[1]=y
↦ _[2]=z
  delete(maxideal(1),1..2);
↦ _[1]=z
```

See Section 4.6 [ideal], page 80; Section 5.1.62 [insert], page 200; Section 4.9 [intvec], page 93; Section 4.11 [list], page 103; Section 4.14 [module], page 112.

5.1.22 denominator

Syntax: `denominator` (number_expression)

Type: number

Purpose: returns the denominator of a number.

Example:

```
  ring r = 0, x, dp;
  number n = 3/2;
  denominator(n);
↦ 2
```

See Section 5.1.9 [cleardenom], page 163; Section D.2.8.14 [content], page 894; Section 5.1.108 [numerator], page 233.

5.1.23 det

Syntax: `det` (intmat_expression)
`det` (matrix_expression)
`det` (smatrix_expression)
`det` (matrix_expression , string_expression)
`det` (smatrix_expression , string_expression)

Type: int, resp. poly

Purpose: returns the determinant of a square matrix. The applied algorithms depend on type of input or the optional second argument.

The optional second argument specifies the algorithm to use. Possible values are "Bareiss", "SBareiss", "Mu" and "Factory".

Example:

```
  ring r=7,(x,y),wp(2,3);
  matrix m[3][3]=1,2,3,4,5,6,7,8,x;
  det(m);
↦ -3x-1
```

See Section 4.8 [intmat], page 90; Section 4.13 [matrix], page 108; Section 5.1.92 [minor], page 221.

5.1.24 diff

Syntax: `diff (` poly_expression`,` ring_variable `)`
`diff (` vector_expression`,` ring_variable `)`
`diff (` ideal_expression`,` ring_variable `)`
`diff (` module_expression`,` ring_variable `)`
`diff (` matrix_expression`,` ring_variable `)`

Type: the same as the type of the first argument

Syntax: `diff (` ideal_expression`,` ideal_expression `)`

Type: matrix

Syntax: `diff (` number_expression`,` ring_parameter `)`

Type: number

Purpose: computes the partial derivative of a polynomial object by a ring variable (first forms) respectively differentiates each polynomial (1..n) of the second ideal by the differential operator corresponding to each polynomial (1..m) in the first ideal, producing an m x n matrix.
respectively if the coefficient ring is a transcendental field extension, differentiates a number (that is, a rational function) by a transcendental variable (ring parameter).

Example:

```
  ring r=0,(x,y,z),dp;
  poly f=2x3y+3z5;
  diff(f,x);
↦ 6x2y
  vector v=[f,y2+z];
  diff(v,z);
↦ 15z4*gen(1)+gen(2)
  ideal j=x2-yz,xyz;
  ideal i=x2,x2+yz,xyz;
  // corresponds to differential operators
  // d2/dx2, d2/dx2+d2/dydz, d3/dxdydz:
  print(diff(i,j));
↦ 2,0,
↦ 1,x,
↦ 0,1
  // differentiation of rational functions:
  ring R=(0,t),(x),dp;
  number f = t^2/(1-t)^2;
  diff(f,t);
↦ (-2t)/(t3-3t2+3t-1)
```

See Section 5.1.13 [contract], page 167; Section 4.6 [ideal], page 80; Section 5.1.66 [jacob], page 203; Section 4.13 [matrix], page 108; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 5.1.165 [var], page 285; Section 4.23 [vector], page 134.

5.1.25 dim

Syntax:
`dim` (ideal_expression)
`dim` (module_expression)
`dim` (resolution_expression)
`dim` (ideal_expression , ideal_expression)
`dim` (module_expression , ideal_expression)

Type: int

Purpose: computes the dimension of the ideal, resp. module, generated by the leading monomials of the given generators of the ideal, resp. module. This is also the dimension of the ideal if it is represented by a standard basis.
`dim(I,J)` is the dimension of `I/J`.
`dim(` res `)` computes the cohomological dimension of res[1].

Note: The dimension of an ideal I means the Krull dimension of the basering modulo I.
The dimension of a module is the dimension of its annihilator ideal.
In the case of ideal (1), -1 is returned.

Example:

```
  ring r=32003,(x,y,z),dp;
  ideal I=x2-y,x3;
  dim(std(I));
↦ 1
  dim(std(ideal(1)));
↦ -1
```

See Section 5.1.20 [degree], page 171; Section 4.6 [ideal], page 80; Section 5.1.101 [mult], page 228; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.26 division

Syntax:
`division` (ideal_expression, ideal_expression)
`division` (module_expression, module_expression)
`division` (ideal_expression, ideal_expression, int_expression)
`division` (module_expression, module_expression, int_expression)
`division` (ideal_expression, ideal_expression, int_expression, intvec_expression)
`division` (module_expression, module_expression, int_expression, intvec_expression)

Type: list

Purpose: `division` computes a division with remainder. For two ideals resp. modules `M` (first argument) and `N` (second argument), it returns a list `T,R,U` where `T` is a matrix, `R` is an ideal resp. a module, and `U` is a diagonal matrix of units such that `matrix(M)*U=matrix(N)*T+matrix(R)` is a standard representation for the normal form `R` of `M` with respect to a standard basis of `N`. `division` uses different algorithms depending on whether `N` is represented by a standard basis. For a polynomial basering, the matrix `U` is the identity matrix. A matrix `T` as above is also computed by `lift`.

For additional arguments `n` (third argument) and `w` (fourth argument), `division` returns a list `T,R` as above such that `matrix(M)=matrix(N)*T+matrix(R)` is a standard representation for the normal form `R` of `M` with respect to `N` up to weighted degree `n` with respect to the weight vector `w`. The weighted degree of `T` and `R` respect to `w` is at most `n`. If the weight vector `w` is not given, `division` uses the standard weight vector `w=1,...,1`.

Example:

```
ring R=0,(x,y),ds;
poly f=x5+x2y2+y5;
division(f,jacob(f)); // automatic conversion: poly -> ideal
↦ [1]:
↦    _[1,1]=1/5x
↦    _[2,1]=3/10y
↦ [2]:
↦    _[1]=-1/2y5
↦ [3]:
↦    _[1,1]=1
division(f^2,jacob(f));
↦ [1]:
↦    _[1,1]=1/20x6-9/80xy5-5/16x7y+5/8x2y6
↦    _[2,1]=1/8x2y3+1/5x5y+1/20y6-3/4x3y4-5/4x6y2-5/16xy7
↦ [2]:
↦    _[1]=0
↦ [3]:
↦    _[1,1]=1/4-25/16xy
division(ideal(f^2),jacob(f),10);
↦ // ** _ is no standard basis
↦ [1]:
↦    _[1,1]=-75/8y9
↦    _[2,1]=1/2x2y3+x5y-1/4y6-3/2x3y4+15/4xy7+375/16x2y8
↦ [2]:
↦    _[1]=x10+9/4y10
```

See Section 4.6 [ideal], page 80; Section 5.1.80 [lift], page 211; Section 4.14 [module], page 112; Section 4.17.3 [poly operations], page 121; Section 5.1.131 [reduce], page 251.

5.1.27 dump

Syntax: `dump` (link_expression)

Type: none

Purpose: dumps (i.e., writes in a "message" or "block") the state of the Singular session (i.e., all defined variables and their values) to the specified link (which must be either an ASCII or ssi link) such that a `getdump` can retrieve it later on.

Example:

```
ring r;
// write the whole session to the file dump.ascii
// in ASCII format
dump(":w dump.ascii");
kill r;                  // kill the basering
// reread the session from the file
```

```
  // redefining everything which was not explicitly killed before
  getdump("dump.ascii");
  r;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

Restrictions:

For ASCII links, integer matrices contained in lists are dumped as integer list elements (and not as integer matrices), and lists of lists are dumped as one flatted list. Furthermore, links themselves are not dumped.

See Section 5.1.52 [getdump], page 191; Section 4.10 [link], page 96; Section 5.1.174 [write], page 289.

5.1.28 eliminate

Syntax: `eliminate` (ideal_expression, product_of_ring_variables)
`eliminate` (module_expression, product_of_ring_variables)
`eliminate` (ideal_expression, intvec_expression)
`eliminate` (module_expression, intvec_expression)
`eliminate` (ideal_expression, product_of_ring_variables, bigintvec_hilb)
`eliminate` (module_expression, product_of_ring_variables, bigintvec_hilb)

Type: the same as the type of the first argument

Purpose: eliminates variables occurring as factors/entries of the second argument from an ideal (resp. a submodule of a free module), by intersecting it (resp. each component of the submodule) with the subring not containing these variables.
`eliminate` does not need a special ordering nor a standard basis as input.

Note: Since elimination is expensive, for homogeneous input it might be useful first to compute the Hilbert function of the ideal (first argument) with a fast ordering (e.g., `dp`). Then make use of it to speed up the computation: a Hilbert-driven elimination uses the intvec provided as the third argument.
If the ideal (resp. module) is not homogeneous with weights 1, this intvec will be silently ignored.

Example:

```
  ring r=32003,(x,y,z),dp;
  ideal i=x2,xy,y5;
  eliminate(i,x);
↦ _[1]=y5
  ring R=0,(x,y,t,s,z),dp;
  ideal i=x-t,y-t2,z-t3,s-x+y3;
  eliminate(i,ts);
↦ _[1]=y2-xz
↦ _[2]=xy-z
↦ _[3]=x2-y
  ideal j=x2,xy,y2;
  intvec v=hilb(std(j),1);
  eliminate(j,y,v);
↦ _[1]=x2
```

See Section 5.1.56 [hilb], page 195; Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271.

5.1.29 eval

Syntax: `eval` (expression)

Type: none

Purpose: evaluates (quoted) expressions. Within a quoted expression, the quote can be "undone" by an `eval` (i.e., each eval "undoes" the effect of exactly one quote). Used only when receiving a quoted expression from an ssi link, with `quote` and `write` to prevent local evaluations when writing to an ssi link.

Example:

```
  link l="ssi:w example.ssi";
  ring r=0,(x,y,z),ds;
  ideal i=maxideal(3);
  ideal j=x7+x3,x2,z;
  // compute i+j before writing, but not std
  // this writes 'std(ideal(x3,...,z))'
  write (l, quote(std(eval(i+j))));
  option(prot);
  close(l);
  // now read it in again and evaluate
  // read(l) forces to compute 'std(ideal(x3,...,z))'
  read(l);
↦ _[1]=z
↦ _[2]=x2
↦ _[3]=xy2
↦ _[4]=y3
  close(l);
```

See Section 4.10.5 [Ssi links], page 99; Section 5.1.126 [quote], page 247; Section 5.1.174 [write], page 289.

5.1.30 ERROR

Syntax: `ERROR` (string_expression)

Type: none

Purpose: Immediately interrupts the current computation, returns to the top-level, and displays the argument `string_expression` as error message.

Note: This should be used as an emergency, resp. failure, exit within procedures.

Example:

```
int i=1;
proc myError() {ERROR("Need to leave now");i=2;}
myError();
↦    ? Need to leave now
↦    ? leaving ::myError (0)
i;
↦ 1
```

5.1.31 example

Syntax: `example` topic ;

Purpose: computes an example for `topic`. Examples are available for all SINGULAR kernel and library functions. Where available (e.g., within Emacs), use `<TAB>` completion for a list of all available example `topics`.

Example:

```
example prime;
example intvec_declarations;
```

Section 5.1.54 [help], page 193

5.1.32 execute

Syntax: `execute (` string_expression `)`

Type: none

Purpose: executes a string containing a sequence of SINGULAR commands.

Note: The command `return` cannot appear in the string.
`execute` should be avoided in procedures whenever possible, since it may give rise to name conflicts. Moreover, such procedures cannot be precompiled (a feature which SINGULAR will provide in the future).

Example:

```
  ring r=32003,(x,y,z),dp;
  ideal i=x+y,z3+22y;
  write(":w save_i",i);
  ring r0=0,(x,y,z),Dp;
  string s="ideal k="+read("save_i")+";";
  s;
↦ ideal k=x+y,z3+22y;
  execute(s); // define the ideal k
  k;
↦ k[1]=x+y
↦ k[2]=z3+22y
```

5.1.33 extgcd

Syntax: `extgcd (` int_expression`,` int_expression `)`
`extgcd (` bigint_expression`,` bigint_expression `)`
`extgcd (` poly_expression`,` poly_expression `)`

Type: list of 3 objects of the same type as the type of the arguments

Purpose: computes extended gcd: the first element is the greatest common divisor of the two arguments, the second and third are factors such that if `list L=extgcd(a,b);` then L[1]=a*L[2]+b*L[3].

Note: Polynomials must be univariate (in the same variable) to apply `extgcd`.

Example:

```
  extgcd(24,10);
↦ [1]:
↦    2
↦ [2]:
↦    -2
↦ [3]:
↦    5
  ring r=0,(x,y),lp;
  extgcd(x4-x6,(x2+x5)*(x2+x3));
↦ [1]:
↦    x5+x4
↦ [2]:
↦    1/2x2+1/2x+1/2
↦ [3]:
↦    1/2
```

See Section 5.1.50 [gcd], page 190; Section 4.7 [int], page 85.

5.1.34 facstd

Syntax: `facstd` (ideal_expression)
`facstd` (ideal_expression, ideal_expression)

Type: list of ideals

Purpose: returns a list of ideals computed by the factorizing Groebner basis algorithm.
The intersection of these ideals has the same zero-set as the input, i.e., the radical of the intersection coincides with the radical of the input ideal. In many (but not all!) cases this is already a decomposition of the radical of the ideal. (Note however that in general, no inclusion between the input and output ideals holds.)
The second, optional argument gives a list of polynomials which define non-zero constraints: those ideals which contain one of the constraint polynomials are omitted from the output list. Thus the zero set of the intersection of the output ideals is contained in the zero set V of the first input ideal and contains the complement in V of the zero set of the second input ideal.

Note: Not implemented for baserings over real ground fields and Galois fields (that is, only implemented for ground fields for which Section 5.1.36 [factorize], page 180 is implemented).

Example:

```
  ring r=32003,(x,y,z),(c,dp);
  ideal I=xyz,x2z;
  facstd(I);
↦ [1]:
↦    _[1]=z
↦ [2]:
↦    _[1]=x
  facstd(I,x);
↦ [1]:
↦    _[1]=z
```

See Section 4.6 [ideal], page 80; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271.

5.1.35 factmodd

Syntax: `factmodd (` poly_expression, int_expression
`[,` poly_expression, poly_expression `]`
`[,` int_expression, int_expression `]`
`)`

Type: list of polys

Purpose: Computes a factorization of a polynomial h(x, y) in K[[x]][y] up to a certain degree in x, whenever a factorization of h(0, y) is provided or can be computed.
The algorithm is based on Hensel's lemma: Let h(x, y) denote a monic polynomial in y of degree m + n with coefficients in K[[x]]. Suppose there are two monic factors f_0(y) (of degree n) and g_0(y) of degree (m) such that
h(0, y) = f_0(y) * g_0(y) and <f_0, g_0> = K[y].
Fix an integer d >= 0. Then there are monic polynomials in y with coefficients in K[[x]], namely f(x, y) of degree n and g(x, y) of degree m such that
h(x, y) = f(x, y) * g(x, y) modulo <x^(d+1)> (*).
The function's six arguments are h, d, f_0, g_0, xIndex, and yIndex, where xIndex and yIndex denote indices of ring variables that are to play the roles of x and y as above.
h must be provided as an element of K[x,y] since all terms of h with x-degree larger than d can be ignored due to (*).
If f_0 and g_0 are not given, the algorithm computes the factorization of h(0, y) and is expected to find exactly two distinct factors (which may appear with multiplicities larger than 1) and uses these as f_0 and g_0.
If xIndex and yIndex are missing they will be expected to be 1 and 2, respectively.

Note: The function expects the ground ring to contain at least two variables.

Example:

```
  ring r = 0, (x,y), dp;
  poly f0 = y240; poly g0 = y102+1;
  poly h = y342+14x260+7x140y110+2x120y130+y240;
  int d = 260;
  list L = factmodd(h, d, f0, g0); L;
↦ [1]:
↦    -14x260y204-4x240y224-14x260y102-7x140y212-2x120y232+14x260+7x140y110
   2x120y130+y240
↦ [2]:
↦    42x260y66+8x240y86+7x140y74+2x120y94+y102+1
  // check result: next output should be zero
  reduce(h - L[1] * L[2], std(x^(d+1)));
↦ 0
```

See Section 5.1.36 [factorize], page 180.

5.1.36 factorize

Syntax: `factorize (` poly_expression `)`
`factorize (` poly_expression`, 0 )`
`factorize (` poly_expression`, 2 )`

Type: list of ideal and intvec

Syntax: `factorize (` poly_expression`, 1 )`

Type: ideal

Purpose: computes the irreducible factors (as an ideal) of the polynomial together with or without the multiplicities (as an intvec) depending on the second argument:

0: returns factors and multiplicities, first factor is a constant.
May also be written with only one argument.
1: returns non-constant factors (no multiplicities).
2: returns non-constant factors and multiplicities.

Note: Not implemented for the coefficient fields real, finite fields of type `(p^n,a)` and ZZ/m.

Example:

```
  ring r=32003,(x,y,z),dp;
  factorize(9*(x-1)^2*(y+z));
↦ [1]:
↦    _[1]=9
↦    _[2]=y+z
↦    _[3]=x-1
↦ [2]:
↦    1,1,2
  factorize(9*(x-1)^2*(y+z),1);
↦ _[1]=y+z
↦ _[2]=x-1
  factorize(9*(x-1)^2*(y+z),2);
↦ [1]:
↦    _[1]=y+z
↦    _[2]=x-1
↦ [2]:
↦    1,2
  ring rQ=0,x,dp;
  poly f = x2+1;            // irreducible in Q[x]
  factorize(f);
↦ [1]:
↦    _[1]=1
↦    _[2]=x2+1
↦ [2]:
↦    1,1
  ring rQi = (0,i),x,dp;
  minpoly = i2+1;
  poly f = x2+1;            // splits into linear factors in Q(i)[x]
  factorize(f);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+(-i)
↦    _[3]=x+(i)
↦ [2]:
↦    1,1,1
```

See Section D.4.1.1 [absFactorize], page 1001; Section 4.17 [poly], page 120.

5.1.37 farey

Syntax: `farey` (bigint_expression , bigint_expression)
`farey` (ideal_expression , bigint_expression)
`farey` (module_expression , bigint_expression)
`farey` (matrix_expression , bigint_expression)
`farey` (smatrix_expression , bigint_expression)
`farey` (list_expression , bigint_expression)

Type: type of the first argument (unless it is `list`)

Purpose: lift the first argument modulo the second to the rationals.
The (coefficients of the) result a/b is the best approximation under the condition $|a|, |b| <= \sqrt{(N-1)/2}$ `farey(list(a,b,..),B)` is equivalent to `list(farey(a,B),farey(b,B),...)`.

Note: The current coefficient field must be the rationals.

Example:

```
ring r=0,x,dp;
farey(2,32003);
↦ 2
```

See Section 5.1.8 [chinrem], page 162.

5.1.38 fetch

Syntax: `fetch` (ring_name, name)
`fetch` (ring_name, name, intvec_expression)
`fetch` (ring_name, name, intvec_expression, intvec_expression)

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: maps objects between rings. `fetch` is the identity map between rings and qrings, in the first case the i-th variable of the source ring is mapped to the i-th variable of the basering. If the basering has less variables than the source ring these variables are mapped to zero. In the 3rd and 4th arguments the intvec describes the permutation of the variables: an i at position j maps the variable `var(j)` of the source to the variable `var(i)` of the destination. Negative numbers (and the fourth argument) describe mapping of parameters.
A zero means that that variable/parameter is mapped to 0.
The coefficient fields must be compatible. (See Section 4.12 [map], page 106 for a description of possible mappings between different ground fields).
`fetch` offers a convenient way to change variable names or orderings, or to map objects from a ring to a quotient ring of that ring or vice versa.
`option(Imap);` reports the mapping.

Note: Compared with `imap`, `fetch` uses the position of the ring variables, not their names.

Example:

```
ring r=0,(x,y,z),dp;
ideal i=maxideal(2);
ideal j=std(i);
poly f=x+y2+z3;
vector v=[f,1];
qring q=j;
poly f=fetch(r,f);
```

```
  f;
↦ z3+y2+x
  vector v=fetch(r,v);
  v;
↦ z3*gen(1)+y2*gen(1)+x*gen(1)+gen(2)
  ideal i=fetch(r,i);
  i;
↦ i[1]=z2
↦ i[2]=yz
↦ i[3]=y2
↦ i[4]=xz
↦ i[5]=xy
↦ i[6]=x2
  ring rr=0,(a,b,c),lp;
  poly f=fetch(q,f);
  f;
↦ a+b2+c3
  vector v=fetch(r,v);
  v;
↦ a*gen(1)+b2*gen(1)+c3*gen(1)+gen(2)
  ideal k=fetch(q,i);
  k;
↦ k[1]=c2
↦ k[2]=bc
↦ k[3]=b2
↦ k[4]=ac
↦ k[5]=ab
↦ k[6]=a2
  fetch(q,i,1..nvars(q)); // equivalent to fetch(q,i)
↦ _[1]=c2
↦ _[2]=bc
↦ _[3]=b2
↦ _[4]=ac
↦ _[5]=ab
↦ _[6]=a2
```

See Section 5.1.59 [imap], page 198; Section 4.12 [map], page 106; Section 5.1.111 [option], page 234; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127.

5.1.39 fglm

Syntax: `fglm` (ring_name, ideal_name)

Type: ideal

Purpose: computes for the given ideal in the given ring a reduced Groebner basis in the current ring, by applying the so-called FGLM (Faugere, Gianni, Lazard, Mora) algorithm.
The main application is to compute a lexicographical Groebner basis from a reduced Groebner basis with respect to a degree ordering. This can be much faster than computing a lexicographical Groebner basis directly.

Assume: The ideal must be zero-dimensional and given as a reduced Groebner basis in the given ring. The monomial ordering must be global.

Note: The only permissible differences between the given ring and the current ring are the monomial ordering and a permutation of the variables, resp. parameters.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal i=y3+x2, x2y+x2, x3-x2, z4-x2-y;
  option(redSB);   // force the computation of a reduced SB
  i=std(i);
  vdim(i);
↦ 28
  ring s=0,(z,x,y),lp;
  ideal j=fglm(r,i);
  j;
↦ j[1]=y4+y3
↦ j[2]=xy3-y3
↦ j[3]=x2+y3
↦ j[4]=z4+y3-y
```

See Section 5.1.40 [fglmquot], page 184; Section 5.1.111 [option], page 234; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section 5.1.168 [vdim], page 286.

5.1.40 fglmquot

Syntax: `fglmquot` (ideal_expression, poly_expression)

Type: ideal

Purpose: computes a reduced Groebner basis of the ideal quotient `I:p` of a zero-dimensional ideal `I` and a polynomial `p` using FGLM-techniques.

Assume: The ideal must be zero-dimensional and given as a reduced Groebner basis in the given ring. The polynomial must be reduced with respect to the ideal.

Example:

```
  ring r=0,(x,y,z),lp;
  ideal i=y3+x2,x2y+x2,x3-x2,z4-x2-y;
  option(redSB);   // force the computation of a reduced SB
  i=std(i);
  poly p=reduce(x+yz2+z10,i);
  ideal j=fglmquot(i,p);
  j;
↦ j[1]=z12
↦ j[2]=yz4-z8
↦ j[3]=y2+y-z8-z4
↦ j[4]=x+y-z10-z6-z4
```

See Section 5.1.39 [fglm], page 183; Section 5.1.111 [option], page 234; Section 5.1.127 [quotient], page 248; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.41 files, input from

Syntax: `<` "filename"

Type: none

Purpose: Read and execute the content of the file filename. Shorthand for `execute(read(filename))`.

Example:

```
< "example"; //read in the file example and execute it
```

See Section 5.1.32 [execute], page 178; Section 5.1.130 [read], page 250.

5.1.42 find

Syntax: `find` (string_expression, substring_expression)
`find` (string_expression, substring_expression, int_expression)

Type: int

Purpose: returns the first position of the substring in the string or 0 (if not found), starts the search at the position given in the 3rd argument.

Example:

```
  find("Aac","a");
↦ 2
  find("abab","a"+"b");
↦ 1
  find("abab","a"+"b",2);
↦ 3
  find("abab","ab",3);
↦ 3
  find("0123","abcd");
↦ 0
```

See Section 4.22 [string], page 130.

5.1.43 finduni

Syntax: `finduni` (ideal_expression)

Type: ideal

Purpose: returns an ideal which is contained in the ideal_expression, such that the i-th generator is a univariate polynomial in the i-th ring variable.
The polynomials have minimal degree w.r.t. this property.

Assume: The ideal must be zero-dimensional and given as a reduced Groebner basis in the current ring.

Example:

```
  ring  r=0,(x,y,z), dp;
  ideal i=y3+x2,x2y+x2,z4-x2-y;
  option(redSB);  // force computation of reduced basis
  i=std(i);
  ideal k=finduni(i);
  print(k);
↦ x4-x2,
↦ y4+y3,
↦ z12
```

See Section 5.1.111 [option], page 234; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.44 flintQ

Syntax: `flintQ` (list_of_names)

Type: cring

Purpose: returns a coefficient ring of multivariate rational functions over Q to be used in ring definitions. Require flint >=2.5.3.

Example:

```
  LIB "flint.so";
  ring R1=flintQ(a,b),(x,y),dp;
  R1;
↦ // coefficients: flintQQ(a,b)
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
```

See Section 4.1 [cring], page 73; Section 4.20 [ring], page 127.

5.1.45 Float

Syntax: `Float` ( )
`Float` (int_expression)
`Float` (int_expression , int_expression)

Type: cring

Purpose: returns a coefficient ring of floating point (inexact) real number to be used in ring definitions.

Example:

```
  ring R1=Float(),(x,y),dp;
  R1;
↦ // coefficients: Float()
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
  ring R2=Float(10,20),(a,b),dp;
  R2;
↦ // coefficients: Float(10,20)
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    a b
↦ //        block   2 : ordering C
```

See Section 4.1 [cring], page 73; Section 4.20 [ring], page 127.

5.1.46 fprintf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `fprintf` (link_expression, string_expression [, any_expressions])

Return: none

Purpose: `fprintf(l,fmt,...);` performs output formatting. The second argument is a format control string. Additional arguments may be required, depending on the content of the control string. A series of output characters is generated as directed by the control string; these characters are written to the link l. The control string `fmt` is simply text to be copied, except that the string may contain conversion specifications.
Type `help print;` for a listing of valid conversion specifications. As an addition to the conversions of `print`, the `%n` and `%2` conversion specification does not consume an additional argument, but simply generates a newline character.

Note: If one of the additional arguments is a list, then it should be enclosed once more into a `list()` command, since passing a list as an argument flattens the list by one level.

Example:

```
  ring r=0,(x,y,z),dp;
module m=[1,y],[0,x+z];
intmat M=betti(mres(m,0));
list l=r,m,M;
link li="";   // link to stdout
fprintf(li,"s:%s,l:%l",1,2);
↦ s:1,l:int(2)
fprintf(li,"s:%s",l);
↦ s:(QQ),(x,y,z),(dp(3),C)
fprintf(li,"s:%s",list(l));
↦ s:(QQ),(x,y,z),(dp(3),C),y*gen(2)+gen(1),x*gen(2)+z*gen(2),1,1
fprintf(li,"2l:%2l",list(l));
↦ 2l:list("(QQ),(x,y,z),(dp(3),C)",
↦ module(y*gen(2)+gen(1),
↦ x*gen(2)+z*gen(2)),
↦ intmat(intvec(1,1 ),1,2))
↦
fprintf(li,"%p",list(l));
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=y*gen(2)+gen(1)
↦    _[2]=x*gen(2)+z*gen(2)
↦ [3]:
↦    1,1
fprintf(li,"%;",list(l));
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=y*gen(2)+gen(1)
↦    _[2]=x*gen(2)+z*gen(2)
↦ [3]:
```

```
↦    1,1
↦
fprintf(li,"%b",M);
↦            0     1
↦ ------------------
↦      0:     1     1
↦ ------------------
↦ total:     1     1
↦
```

See also: Section 5.1.120 [print], page 242; Section 5.1.121 [printf], page 244; Section 5.1.148 [sprintf], page 267; Section 4.22 [string], page 130.

5.1.47 freemodule

Syntax: `freemodule` (int_expression)

Type: module

Purpose: creates the free module of rank n generated by `gen(1)`, ..., `gen(n)`.

Example:

```
  ring r=32003,(x,y),(c,dp);
  freemodule(3);
↦ _[1]=[1]
↦ _[2]=[0,1]
↦ _[3]=[0,0,1]
  matrix m=freemodule(3); // generates the 3x3 unit matrix
  print(m);
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
```

See Section 5.1.51 [gen], page 191; Section 4.14 [module], page 112.

5.1.48 fres

Syntax: `fres` (ideal_expression/module_expression , int_expression , [string_expression])

Type: resolution

Purpose: computes a (not necessarily minimal) free resolution of the input ideal/module, using Schreyer's algorithm, see reference.
If the second argument is `n > 0`, then the resolution is computed up to step `n`. If it is `0`, `fres` computes the whole resolution.
The optional third argument can be set to

- `"complete"` (default) to compute the whole syzygy module in each step,
- `"frame"` to compute only the so-called frame,
- `"extended frame"` to compute only the first two terms of each generator w.r.t. the induced monomial ordering, or
- `"single module"` to return only the frame of each module except the last one and to return the last module in its entirety. This option can be used to reduce the amount of memory needed for the computation.

Note: The input ideal/module must be a standard basis.

Reference:

B. Erocal, O. Motsak, F.-O. Schreyer, A. Steenpass: Refined Algorithms to Compute Syzygies. J. Symb. Comput. 74 (2016), 308-327. `http://arxiv.org/abs/1502.01654`

Example:

```
  ring r = 0, (w,x,y,z), dp;
  ideal I = w2-xz, wx-yz, x2-wy, xy-z2, y2-wz;
  attrib(I, "isSB", 1);
  resolution s = fres(I, 0);
  s;
↦  1      5      6      2
↦ r <--   r <--   r <--   r
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
  print(betti(s, 0), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     5     5     1
↦     2:     -     -     1     1
↦ ------------------------------
↦ total:     1     5     6     2
↦
  list l = s;
  print(l[1]);
↦ w2-xz,
↦ wx-yz,
↦ x2-wy,
↦ xy-z2,
↦ y2-wz
  print(l[2]);
↦ -x,y, 0, -z,0, -y2+wz,
↦ w, -x,-y,0, z, z2,
↦ -z,w, 0, -y,0, 0,
↦ 0, 0, w, x, -y,-yz,
↦ 0, 0, -z,-w,x, w2
  print(l[3]);
↦ 0, -y2+wz,
↦ y, z2,
↦ -x,-wy,
↦ w, yz,
↦ -z,-w2,
↦ 1, x
```

See Section A.3.4 [Free resolution], page 719; Section 5.1.93 [minres], page 223; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269; Section 5.1.156 [syz], page 280.

5.1.49 frwalk

Syntax: `frwalk` (ring_name, ideal_name)
`frwalk` (ring_name, ideal_name , int_expression)

Type: ideal

Purpose: computes for the ideal `ideal_name` in the ring `ring_name` a Groebner basis in the current ring, by applying the fractal walk algorithm.
The main application is to compute a lexicographical Groebner basis from a reduced Groebner basis with respect to a degree ordering. This can be much faster than computing a lexicographical Groebner basis directly.

Note: When calling `frwalk`, the only permissible difference between the ring `ring_name` and the active base ring is the monomial ordering.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal i=y3+x2, x2y+x2, x3-x2, z4-x2-y;
  i=std(i);
  ring s=0,(x,y,z),lp;
  ideal j=frwalk(r,i);
  j;
↦ j[1]=z12
↦ j[2]=yz4-z8
↦ j[3]=y2+y-z8-z4
↦ j[4]=xy-xz4-y+z4
↦ j[5]=x2+y-z4
```

See Section 5.1.39 [fglm], page 183; Section 5.1.53 [groebner], page 191; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271.

5.1.50 gcd

Syntax: `gcd (` int_expression`,` int_expression `)`
`gcd (` bigint_expression`,` bigint_expression `)`
`gcd (` number_expression`,` number_expression `)`
`gcd (` poly_expression`,` poly_expression `)`

Type: the same as the type of the arguments

Purpose: computes the greatest common divisor.

Note: Not implemented for the coefficient fields real and finite fields of type `(p^n,a)`.
The gcd of two numbers is their gcd as integer numbers or polynomials, otherwise it is not defined.

Example:

```
  gcd(2,3);
↦ 1
  gcd(bigint(2)^20,bigint(3)^23);     // also applicable for bigints
↦ 1
  typeof(_);
↦ bigint
  ring r=0,(x,y,z),lp;
  gcd(3x2*(x+y),9x*(y2-x2));
↦ x2+xy
  gcd(number(6472674604870),number(878646537247372));
↦ 2
```

See Section 4.2 [bigint], page 74; Section 5.1.33 [extgcd], page 178; Section 4.7 [int], page 85; Section 4.15 [number], page 115.

5.1.51 gen

Syntax: `gen` (int_expression)

Type: vector

Purpose: returns the i-th free generator of a free module.

Example:

```
  ring r=32003,(x,y,z),(c,dp);
  gen(3);
↦ [0,0,1]
  vector v=gen(5);
  poly f=xyz;
  v=v+f*gen(4); v;
↦ [0,0,0,xyz,1]
  ring rr=32003,(x,y,z),dp;
  fetch(r,v);
↦ xyz*gen(4)+gen(5)
```

See Section 5.1.47 [freemodule], page 188; Section 4.7 [int], page 85; Section 4.23 [vector], page 134.

5.1.52 getdump

Syntax: `getdump` (link_expression)

Type: none

Purpose: reads the content of the entire file, resp. link, and restores all variables from it. For ASCII links, `getdump` is equivalent to an `execute(read(` link `))` command. For ssi links, `getdump` should only be used on data which were previously `dump`'ed.

Example:

```
  int i=3;
  dump(":w example.txt");
  kill i;
  option(noredefine);
  getdump("example.txt");
  i;
↦ 3
```

Restrictions:
`getdump` is not supported for DBM links, or for a link connecting to `stdin` (standard input).

See Section 5.1.27 [dump], page 175; Section 4.10 [link], page 96; Section 5.1.130 [read], page 250.

5.1.53 groebner

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `groebner` (ideal_expression)
`groebner` (module_expression)
`groebner` (ideal_expression, list of string_expressions)
`groebner` (ideal_expression, list of string_expressions and int_expression)

Type: type of the first argument

Purpose: computes a standard basis of the first argument `I` (ideal or module) by a heuristically chosen method (default) or by a method specified by further arguments of type string. Possible methods are:
- the direct methods `"std"` or `"slimgb"` without conversion,
- conversion methods `"hilb"` or `"fglm"` where a Groebner basis is first computed with an "easy" ordering and then converted to the ordering of the basering by the Hilbert driven Groebner basis computation or by linear algebra. The actual computation of the Groebner basis can be specified by `"std"` or by `"slimgb"` (not for all orderings implemented).
- `"HC"`: using the "high corner" from char p in char 0, finding a SB for 0-dimensional ideals in local orderings faster.
A further string `"par2var"` converts parameters to an extra block of variables before a Groebner basis computation (and afterwards back). `option(prot)` informs about the chosen method.

Hint: Since there exists no uniform best method for computing standard bases, and since the difference in performance of a method on different examples can be huge, it is recommended to test, for hard examples, first various methods on a simplified example (e.g. use characteristic 32003 instead of 0 or substitute a subset of parameters/variables by integers, etc.).

Example:

```
  intvec opt = option(get);
option(prot);
ring r  = 0,(a,b,c,d),dp;
ideal i = a+b+c+d,ab+ad+bc+cd,abc+abd+acd+bcd,abcd-1;
groebner(i);
↦ std in (QQ),(a,b,c,d),(dp(4),C)
↦ [65535:2]1(3)s2(2)s3s4-s5ss6-s7--
↦ product criterion:8 chain criterion:5
↦ _[1]=a+b+c+d
↦ _[2]=b2+2bd+d2
↦ _[3]=bc2+c2d-bd2-d3
↦ _[4]=bcd2+c2d2-bd3+cd3-d4-1
↦ _[5]=bd4+d5-b-d
↦ _[6]=c3d2+c2d3-c-d
↦ _[7]=c2d4+bc-bd+cd-2d2
ring s  = 0,(a,b,c,d),lp;
ideal i = imap(r,i);
groebner(i,"hilb");
↦ compute hilbert series with std in ring (QQ),(a,b,c,d,@),(dp(5),C)
↦ weights used for hilbert series: 1,1,1,1,1
↦ [1048575:2]1(3)s2(2)s3s4-s5ss6-s7--
↦ product criterion:8 chain criterion:5
↦ std with hilb in (QQ),(a,b,c,d,@),(lp(4),dp(1),C)
↦ [1048575:2]1(6)s2(5)s3(4)s4-s5sshh6(3)shhhhh8shh
↦ product criterion:9 chain criterion:8
↦ hilbert series criterion:9
↦ dehomogenization
↦ simplification
↦ imap to ring (QQ),(a,b,c,d),(lp(4),C)
↦ _[1]=c2d6-c2d2-d4+1
↦ _[2]=c3d2+c2d3-c-d
↦ _[3]=bd4-b+d5-d
↦ _[4]=bc-bd5+c2d4+cd-d6-d2
↦ _[5]=b2+2bd+d2
↦ _[6]=a+b+c+d
ring R  = (0,a),(b,c,d),lp;
minpoly = a2+1;
ideal i = a+b+c+d,ab+ad+bc+cd,abc+abd+acd+bcd,d2-c2b2;
groebner(i,"par2var","slimgb");
↦ //add minpoly to input
↦ compute hilbert series with slimgb in ring (QQ),(b,c,d,a,@),(dp(5),C)
↦ weights used for hilbert series: 1,1,1,1,1
↦ slimgb in ring (QQ),(b,c,d,a,@),(dp(5),C)
↦ CC2M[2,2](2)C3M[1,1](2)4M[2,e1](2)C5M[2,e2](3)C6M[1,1](0)
↦ NF:8 product criterion:15, ext_product criterion:3
↦ std with hilb in (QQ),(b,c,d,a,@),(lp(3),dp(1),dp(1),C)
↦ [1048575:2]1(7)s2(6)s(5)s3(4)s4-s5sshh6(3)shhhhh
↦ product criterion:15 chain criterion:5
↦ hilbert series criterion:7
↦ dehomogenization
↦ simplification
↦ imap to ring (QQ),(b,c,d,a),(lp(3),dp(1),C)
↦ //simplification
↦ (S:4)rtrtrtr
↦ //imap to original ring
↦ _[1]=d2
↦ _[2]=c+(a)
↦ _[3]=b+c+d+(a)
groebner(i,"fglm");          //computes a reduced standard basis
↦ std in (0,a),(b,c,d),(dp(3),C)
↦ [1048575:2]1(3)s2(2)s3s4-s5ss6-s7
↦ (S:2)--
↦ product criterion:9 chain criterion:1
↦ ..+++--
↦ vdim= 2
↦ ..++-+-
↦ _[1]=d2
↦ _[2]=c+(a)
↦ _[3]=b+d
option(set,opt);
ring Rt = (0,t),(x,y,z),ds;
poly F = y10+(t2)*x7y7+x15+x9y6+(2t)*x6y9+x6y6z3+x5y11+z21;
ideal I = jacob(F);
I=groebner(I,"HC","prot");
↦ computing HC in char 32003
↦ found HC in char 32003: x7y2z38
↦ computing std with HC
```

See also: Section D.4.9 [ffmodstd_lib], page 1080; Section D.4.16 [modstd_lib], page 1146; Section D.4.20 [nfmodstd_lib], page 1174; Section 5.1.145 [slimgb], page 265; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section 5.1.153 [stdhilb], page 273.

5.1.54 help

Syntax: `help;`
`help` topic `;`

Type: none

Purpose: displays online help information for `topic` using the currently set help browser. If no `topic` is given, the title page of the manual is displayed.

Note:

- `?` may be used instead of `help`.
- `topic` can be an index entry of the SINGULAR manual or the name of a (loaded) procedure which has a help section.
- `topic` may contain wildcard characters (i.e., `*` characters).
- If a (possibly "wildcarded") `topic` cannot be found (or uniquely matched) a warning is displayed and no help information is provided.
- If `topic` is the name of a (loaded) procedure whose help section has changed w.r.t. the help available in the manual then, instead of displaying the respective help section of the manual in the help browser, the "newer" help section of the procedure is simply printed to the terminal.
- The browser in which the help information is displayed can be either set with the command-line option `--browser=<browser>` (see Section 3.1.6 [Command line options], page 19), or with the command `system("--browser", "<browser>")`. Use the command `system("browsers");` for a list of all available browsers. See Section 3.1.3 [The online help system], page 15, for more details about help browsers.

Example:

```
help;      // display title page of manual
help ring; // display help for 'ring'
?ringe;    // equivalent to 'help ringe;'
↦ // ** No help for topic 'ringe' (not even for '*ringe*')
↦ // ** Try '?;'       for general help
↦ // ** or  '?Index;'  for all available help topics
?ring*;
↦ //  ** No unique help for 'ring*'
↦ //  ** Try one of
↦ ?Rings and orderings; ?Rings and standard bases; ?ring;
↦ ?ring declarations; ?ring operations; ?ring related functions;
↦ ?ring.lib; ?ring_lib; ?ringtensor; ?ringweights;
help Rings and orderings;
help standard.lib;  // displays help for library 'standard.lib'
```

See Section 3.1.6 [Command line options], page 19; Section 3.8 [Libraries], page 55; Section 3.7.1 [Procedure definition], page 51; Section 3.1.3 [The online help system], page 15; Section 5.1.155 [system], page 275.

5.1.55 highcorner

Syntax: `highcorner (` ideal_expression `)`
`highcorner (` module_expression `)`

Type: poly, resp. vector

Purpose: returns the smallest monomial not contained in the ideal, resp. module, generated by the initial terms of the given generators. If the generators are a standard basis, this is also the smallest monomial not contained in the ideal, resp. module.
If the ideal, resp. module, is not zero-dimensional, 0 is returned.
The command works also in global orderings, but is not very useful there.

Note: Let the ideal I be given by a standard basis. Then `highcorner(I)` returns 0 if and only if `dim(I)>0` or `dim(I)=-1`. Otherwise it returns the smallest monomial m not in I which has the following properties (with x_i the variables of the basering):

- if $x_i > 1$ then x_i does not divide m (hence, m=1 if the ordering is global)
- given any set of generators $f_1, \ldots, f_k$ of I, let f_i' be obtained from f_i by deleting the terms divisible by $x_i \cdot m$ for all i with $x_i < 1$. Then $f_1', \ldots, f_k'$ generate I.

Example:

```
ring r=0,(x,y),ds;
ideal i=x3,x2y,y3;
highcorner(std(i));
↦ xy2
highcorner(std(ideal(1)));
↦ 0
```

See Section 5.1.25 [dim], page 174; Section 5.1.53 [groebner], page 191; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.56 hilb

Syntax: `hilb` (ideal_expression)
`hilb` (module_expression)
`hilb` (ideal_expression, int_expression)
`hilb` (module_expression, int_expression)
`hilb` (ideal_expression, int_expression , intvec_expression)
`hilb` (module_expression, int_expression , intvec_expression)

Type: none (if called with one argument)
intvec (if called with two or three arguments)

Purpose: computes the (weighted) Hilbert series of the base ring R modulo the ideal, resp. R^k modulo the module, defined by the leading terms of the generators of the given ideal, resp. module.
If `hilb` is called with one argument, then the first and second Hilbert series together with some additional information are displayed.
If `hilb` is called with two arguments, then the n-th Hilbert series is returned as an intvec, where n = 1, 2 is the second argument.
If a weight vector w is a given as 3rd argument, then the Hilbert series is computed w.r.t. these weights w (by default all weights are set to 1).

Caution: The last entry of the returned intvec is not part of the actual Hilbert series, but is used in the Hilbert driven standard basis computation (see Section 5.1.153 [stdhilb], page 273). (It is the minimum weight of the module generators or 0).

Syntax: `hilb` (intvec_expression)

Type: intvec

Purpose: computes the second Hilbert series from the first, i.e. if `intvec v=hilb(I,1);` then `hilb(v)` yields the same result as `hilb(I,2)`.

Syntax: `hilb` (ideal_expression`,` ring`,` string_expression)
`hilb` (module_expression`,` ring`,` string_expression)

Type: none

Purpose: computes the (weighted) Hilbert series of the base ring R modulo the ideal, resp. R^k modulo the module, defined by the leading terms of the generators of the given ideal, resp. module.
The series is stored as `poly` under the name give as string in the given ring.

Note: If the input is homogeneous w.r.t. the weights and a standard basis, the result is the (weighted) Hilbert series of the original ideal, resp. module.

Example:

```
  ring Qt=QQ,t,dp;
  ring R=32003,(x,y,z),dp;
  ideal i=x2,y2,z2;
  ideal s=std(i);
  hilb(s);
↦ (-t6+3t4-3t2+1) / (1-t)^3
↦ (t3+3t2+3t+1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 8
  hilb(s,1);
↦ 1,0,-3,0,3,0,-1,0
  hilb(s,2);
↦ 1,3,3,1,0
  intvec w=2,2,2;
  hilb(s,1,w);
↦ 1,0,0,0,-3,0,0,0,3,0,0,0,-1,0
  hilb(s,Qt,"h");
  setring Qt;h;
↦ -t6+3t4-3t2+1
```

See Section C.2 [Hilbert function], page 774; Section 4.6 [ideal], page 80; Section 4.9 [intvec], page 93; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271; Section 5.1.153 [stdhilb], page 273.

5.1.57 homog

Syntax: `homog` (ideal_expression)
`homog` (module_expression)

Type: int

Purpose: tests for homogeneity: returns 1 for homogeneous input, 0 otherwise.

Note: If the current ring has a weighted monomial ordering, `homog` tests for weighted homogeneity w.r.t. the given weights.

Syntax: `homog` (ideal_expression`,` intvec_expression)

Type: int

Purpose: tests for homogeneity wrt. the given weight vector: returns 1 for homogeneous input, 0 otherwise.

Syntax: `homog` (polynomial_expression, ring_variable)
`homog` (vector_expression, ring_variable)
`homog` (ideal_expression, ring_variable)
`homog` (module_expression, ring_variable)

Type: same as first argument

Purpose: homogenizes polynomials, vectors, ideals, or modules by multiplying each monomial with a suitable power of the given ring variable.

Note: If the current ring has a weighted monomial ordering, `homog` computes the weighted homogenization w.r.t. the given weights.
The homogenizing variable must have weight 1.

Example:

```
  ring r=32003,(x,y,z),ds;
  poly s1=x3y2+x5y+3y9;
  poly s2=x2y2z2+3z8;
  poly s3=5x4y2+4xy5+2x2y2z3+y7+11x10;
  ideal i=s1,s2,s3;
  homog(s2,z);
↦ x2y2z4+3z8
  homog(i,z);
↦ _[1]=3y9+x5yz3+x3y2z4
↦ _[2]=x2y2z4+3z8
↦ _[3]=11x10+y7z3+5x4y2z4+4xy5z4+2x2y2z6
  homog(i);
↦ 0
  homog(homog(i,z));
↦ 1
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.58 hres

Syntax: `hres` (ideal_expression, int_expression)

Type: resolution

Purpose: computes a free resolution of an ideal using the Hilbert-driven algorithm.

More precisely, let R be the basering and I be the given ideal. Then `hres` computes a minimal free resolution of R/I

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} R \longrightarrow R/I \longrightarrow 0.$$

If the int_expression k is not zero then the computation stops after k steps and returns a list of modules $M_i = \texttt{module}(A_i)$, i=1..k.

`list L=hres(I,0);` returns a list L of n modules (where n is the number of variables of the basering) such that `L[i]` $= M_i$ in the above notation.

Note: The ideal_expression has to be homogeneous.
Accessing single elements of a resolution may require some partial computations to be finished. Therefore, it may take some time.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=xz,yz,x3-y3;
  def L=hres(I,0);
  print(betti(L),"betti");
↦             0     1     2
↦ ------------------------
↦     0:      1     -     -
↦     1:      -     2     1
↦     2:      -     1     1
↦ ------------------------
↦ total:      1     3     2
↦
  L[2];     // the first syzygy module of r/I
↦ _[1]=-x*gen(1)+y*gen(2)
↦ _[2]=-x2*gen(2)+y2*gen(1)+z*gen(3)
```

See Section 5.1.4 [betti], page 159; Section 5.1.48 [fres], page 188; Section 4.6 [ideal], page 80; Section 4.7 [int], page 85; Section 5.1.83 [lres], page 215; Section 5.1.93 [minres], page 223; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.99 [mres_map], page 226; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.59 imap

Syntax: `imap` (ring_name, name)

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: identity map on common subrings. `imap` is the map between rings and qrings with compatible ground fields which is the identity on variables and parameters of the same name and 0 otherwise. (See Section 4.12 [map], page 106 for a description of possible mappings between different ground fields). Useful for mapping from a homogenized ring to the original ring or for mappings from/to rings with/without parameters. Compared with `fetch`, `imap` uses the names of variables and parameters. Unlike `map` and `fetch` `imap` can map parameters to variables.
Mapping rational functions which are not polynomials to polynomials is undefined (i.e. the result depends on the version).

Example:

```
  ring r=0,(x,y,z,a,b,c),dp;
  ideal i=xy2z3a4b5+1,homog(xy2z3a4b5+1,c); i;
↦ i[1]=xy2z3a4b5+1
↦ i[2]=xy2z3a4b5+c15
  ring r1=0,(a,b,x,y,z),lp;
  ideal j=imap(r,i); j;
↦ j[1]=a4b5xy2z3+1
↦ j[2]=a4b5xy2z3
  ring r2=(0,a,b),(x,y,z),ls;
  ideal j=imap(r,i); j;
↦ j[1]=1+(a4b5)*xy2z3
↦ j[2]=(a4b5)*xy2z3
```

See Section 5.1.38 [fetch], page 182; Section 5.1.57 [homog], page 196; Section 4.12 [map], page 106; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127.

5.1.60 impart

Syntax: `impart` (number_expression)

Type: number

Purpose: returns the imaginary part of a number in a complex ground field,
returns 0 otherwise.

Example:

```
  ring r=(complex,i),x,dp;
  impart(1+2*i);
↦ 2
```

See Section 5.1.133 [repart], page 253.

5.1.61 indepSet

Syntax: `indepSet` (ideal_expression)

Type: intvec

Purpose: computes a maximal set U of independent variables (in the sense defined in the note below) of the ideal given by a standard basis. If `v` is the result then `v[i]` is 1 if and only if the i-th variable of the ring, `x(i)`, is an independent variable. Hence, the set U consisting of all variables `x(i)` with `v[i]=1` is a maximal independent set.

Note: U is a set of independent variables for I if and only if $I \cap K[U] = (0)$, i.e., eliminating the remaining variables gives (0). U is maximal if dim(I)=#U.

Syntax: `indepSet` (ideal_expression, int_expression)

Type: list

Purpose: computes a list of all maximal independent sets of the leading ideal (if the flag is 0), resp. of all those sets of independent variables of the leading ideal which cannot be enlarged.

Example:

```
  ring r=32003,(x,y,u,v,w),dp;
  ideal I=xyw,yvw,uyw,xv;
  attrib(I,"isSB",1);
  indepSet(I);
↦ 1,1,1,0,0
  eliminate(I,vw);
↦ _[1]=0
  indepSet(I,0);
↦ [1]:
↦    1,1,1,0,0
↦ [2]:
↦    0,1,1,1,0
↦ [3]:
↦    1,0,1,0,1
↦ [4]:
↦    0,0,1,1,1
  indepSet(I,1);
↦ [1]:
```

```
↦    1,1,1,0,0
↦ [2]:
↦    0,1,1,1,0
↦ [3]:
↦    1,0,1,0,1
↦ [4]:
↦    0,0,1,1,1
↦ [5]:
↦    0,1,0,0,1
  eliminate(I,xuv);
↦ _[1]=0
```

See Section 4.6 [ideal], page 80; Section 5.1.151 [std], page 271.

5.1.62 insert

Syntax: `insert` (list_expression, expression)
`insert` (list_expression, expression, int_expression)

Type: list

Purpose: inserts a new element (expression) into a list at the beginning, or (if called with 3 arguments) after the given position (the input is not changed).

Example:

```
  list L=1,2;
  insert(L,4,2);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    4
  insert(L,4);
↦ [1]:
↦    4
↦ [2]:
↦    1
↦ [3]:
↦    2
```

See Section 5.1.21 [delete], page 171; Section 4.11 [list], page 103.

5.1.63 interpolation

Syntax: `interpolation` (list, intvec)

Type: ideal

Purpose: `interpolation(l,v)` computes the reduced Groebner basis of the intersection of ideals l[1]^v[1], ..., l[N]^v[N] by applying linear algebra methods.

Assume: Every ideal from the list l must be a maximal ideal of a point and should have the following form: variable_1-coordinate_1, ..., variable_n-coordinate_n, where n is the number of variables in the ring.
The ring should be a polynomial ring over Zp or Q with global ordering.

Example:

```
  ring r=0,(x,y),dp;
  ideal p_1=x,y;
  ideal p_2=x+1,y+1;
  ideal p_3=x+2,y-1;
  ideal p_4=x-1,y+2;
  ideal p_5=x-1,y-3;
  ideal p_6=x,y+3;
  ideal p_7=x+2,y;
  list l=p_1,p_2,p_3,p_4,p_5,p_6,p_7;
  intvec v=2,1,1,1,1,1,1;
  ideal j=interpolation(l,v);
  // generator of degree 3 gives the equation of the unique
  // singular cubic passing
  // through p_1,...,p_7 with singularity at p_1
  j;
↦ j[1]=-4x3-4x2y-2xy2+y3-8x2-4xy+3y2
↦ j[2]=-y4+8x2y+6xy2-2y3+10xy+3y2
↦ j[3]=-xy3+2x2y+xy2+4xy
↦ j[4]=-2x2y2-2x2y-2xy2+y3-4xy+3y2
  // computes values of generators of j at p_4, results should be 0
  subst(j,x,1,y,-2);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
  // computes values of derivatives d/dx of generators at (0,0)
  subst(diff(j,x),x,0,y,0);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
```

See Section 5.1.24 [diff], page 173; Section 5.1.39 [fglm], page 183; Section 5.1.65 [intersect], page 202; Section 5.1.151 [std], page 271; Section 5.1.154 [subst], page 274.

5.1.64 interred

Syntax: `interred` (ideal_expression)
`interred` (module_expression)

Type: the same as the input type

Purpose: interreduces a set of polynomials/vectors.
Input: $f_1, \ldots, f_n$
Output: $g_1, \ldots, g_s$ with $s \leq n$ and the properties

- $(f_1, \ldots, f_n) = (g_1, \ldots, g_s)$,
- $L(g_i) \neq L(g_j)$ for all $i \neq j$,
- in the case of a global ordering (polynomial ring) and `option(redSB);`:
  $L(g_i)$ does not divide m for all monomials m of $\{g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_s\}$,
- in the case of a local ordering (localization of polynomial ring) and `option(redSB);`:
  if $L(g_i)|L(g_j)$ for any $i \neq j$, then $ecart(g_i) > ecart(g_j)$.

Here, $L(g)$ denotes the leading term of g and $ecart(g) := deg(g) - deg(L(g))$.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal i=zx+y3,z+y3,z+xy;
  interred(i);
↦ _[1]=xz-z
↦ _[2]=xy+z
↦ _[3]=y3+xz
  ring R=0,(x,y,z),ds;
  ideal i=zx+y3,z+y3,z+xy;
  interred(i);
↦ _[1]=z+xy
↦ _[2]=xy-y3
↦ _[3]=x2y-y3
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271.

5.1.65 intersect

Syntax: `intersect` (expression_list of ideal_expression)
`intersect` (expression_list of module_expression)

Type: ideal, resp. module

Purpose: computes the intersection of ideals, resp. modules.

Note: If the option `prot` is enabled then the result the used method (elimination/syzygies) is displayed.
An optional last argument specifies the Groebner base algorithm to use. Possible values are `"std"` and `"slimgb"`.

Example:

```
  ring R=0,(x,y),dp;
  ideal i=x;
  ideal j=y;
  intersect(i,j);
↦ _[1]=xy
  ring r=181,(x,y,z),(c,ls);
  ideal id1=maxideal(3);
  ideal id2=x2+xyz,y2-z3y,z3+y5xz;
  ideal id3=intersect(id1,id2,ideal(x,y));
  ideal id4=intersect(id1,id2,"slimgb");
  id3;
↦ id3[1]=yz3+xy6z
↦ id3[2]=yz4-y2z
↦ id3[3]=y2z3-y3
↦ id3[4]=xz3+x2y5z
↦ id3[5]=xyz2+x2z
↦ id3[6]=xy2+x2z2
↦ id3[7]=xy2z+x2y
↦ id3[8]=x2yz+x3
  id4;
↦ id4[1]=xyz2+x2z
↦ id4[2]=xy2z+x2y
↦ id4[3]=x2yz+x3
↦ id4[4]=-yz4+y2z
↦ id4[5]=-y2z3+y3
↦ id4[6]=-xyz3+xy2
↦ id4[7]=z3+xy5z
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.111 [option], page 234.

5.1.66 jacob

Syntax: `jacob` (poly_expression)
`jacob` (ideal_expression)
`jacob` (module_expression)

Type: ideal, if the input is a polynomial
matrix, if the input is an ideal
module, if the input is a module

Purpose: computes the Jacobi ideal, resp. Jacobi matrix, generated by all partial derivatives of the input.

Note: In a ring with n variables, jacob of a module or an ideal (considered as matrix with a single a row) or a polynomial (considered as a matrix with a single entry) is the matrix consisting of horizontally concatenated blocks (in this order): diff(MT,var(1)), ... , diff(MT,var(n)), where MT is the transposed input argument considered as a matrix.

Example:

```
  ring R;
  poly f = x2yz + xy3z + xyz5;
  ideal i = jacob(f); i;
↦ i[1]=yz5+y3z+2xyz
↦ i[2]=xz5+3xy2z+x2z
↦ i[3]=5xyz4+xy3+x2y
  matrix m = jacob(i);
  print(m);
↦ 2yz,        z5+3y2z+2xz, 5yz4+y3+2xy,
↦ z5+3y2z+2xz,6xyz,        5xz4+3xy2+x2,
↦ 5yz4+y3+2xy,5xz4+3xy2+x2,20xyz3
  print(jacob(m));
↦ 0, 2z,        2y,        2z,        6yz,5z4+3y2+2x,2y,        5z4+3y2+2
   20yz3,
↦ 2z,6yz,       5z4+3y2+2x,6yz,       6xz,6xy,       5z4+3y2+2x,6xy,
   20xz3,
↦ 2y,5z4+3y2+2x,20yz3,     5z4+3y2+2x,6xy,20xz3,     20yz3,     20xz3,
   60xyz2
```

See Section 5.1.24 [diff], page 173; Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.109 [nvars], page 233.

5.1.67 janet

Syntax: `janet` (ideal_expression)
`janet` (ideal_expression , int_expression)

Type: ideal

Purpose: computes the Janet basis of the given ideal, resp. the standard basis if 1 is given as the second argument.

Remark: It works only with global orderings.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal i=x*y*z-1,x+y+z,x*y+x*z+y*z; // cyclic 3
  janet(i);
↦ Length of Janet basis: 4
↦ _[1]=x+y+z
↦ _[2]=y2+yz+z2
↦ _[3]=z3-1
↦ _[4]=yz3-y
```

See Section 5.1.53 [groebner], page 191; Section 4.6 [ideal], page 80; Section 5.1.151 [std], page 271.

5.1.68 jet

Syntax: `jet` (poly_expression, int_expression)
`jet` (vector_expression, int_expression)
`jet` (ideal_expression, int_expression)
`jet` (module_expression, int_expression)
`jet` (poly_expression, int_expression, intvec_expression)
`jet` (vector_expression, int_expression, intvec_expression)
`jet` (ideal_expression, int_expression, intvec_expression)
`jet` (module_expression, int_expression, intvec_expression)
`jet` (poly_expression, poly_expression, int_expression, intvec_expression)
`jet` (vector_expression, poly_expression, int_expression, intvec_expression)
`jet` (ideal_expression, matrix_expression, int_expression, intvec_expression)
`jet` (module_expression, matrix_expression, int_expression, intvec_expression)
`jet` (poly_expression, poly_expression, int_expression, intvec_expression)
`jet` (vector_expression, poly_expression, int_expression)
`jet` (ideal_expression, matrix_expression, int_expression)
`jet` (module_expression, matrix_expression, int_expression)

Type: the same as the type of the first argument

Purpose: deletes from the first argument all terms of degree bigger than the second argument.
If a third/fourth argument `w` of type intvec is given, the degree is replaced by the weighted degree defined by `w`.
If a second argument `u` of type poly or matrix is given, the first argument `p` is replaced by `p/u`. In this case, the coefficient must be from a field.

Example:

```
  ring r=32003,(x,y,z),(c,dp);
  jet(1+x+x2+x3+x4,3);
↦ x3+x2+x+1
  poly f=1+x+x2+xz+y2+x3+y3+x2y2+z4;
  jet(f,3);
↦ x3+y3+x2+y2+xz+x+1
  intvec iv=2,1,1;
  jet(f,3,iv);
↦ y3+y2+xz+x+1
```

```
  // the part of f with (total) degree >3:
  f-jet(f,3);
↦ x2y2+z4
  // the homogeneous part of f of degree 2:
  jet(f,2)-jet(f,1);
↦ x2+y2+xz
  // the part of maximal degree:
  jet(f,deg(f))-jet(f,deg(f)-1);
↦ x2y2+z4
  // the absolute term of f:
  jet(f,0);
↦ 1
  // now for other types:
  ideal i=f,x,f*f;
  jet(i,2);
↦ _[1]=x2+y2+xz+x+1
↦ _[2]=x
↦ _[3]=3x2+2y2+2xz+2x+1
  vector v=[f,1,x];
  jet(v,1);
↦ [x+1,1,x]
  jet(v,0);
↦ [1,1]
  v=[f,1,0];
  module m=v,v,[1,x2,z3,0,1];
  jet(m,2);
↦ _[1]=[x2+y2+xz+x+1,1]
↦ _[2]=[x2+y2+xz+x+1,1]
↦ _[3]=[1,x2,0,0,1]
  ring rs=0,x,ds;
  // 1/(1+x) till degree 5
  jet(1,1+x,5);
↦ 1-x+x2-x3+x4-x5
```

See Section 5.1.19 [deg], page 170; Section 4.6 [ideal], page 80; Section 4.7 [int], page 85; Section 4.9 [intvec], page 93; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.69 kbase

Syntax: `kbase` (ideal_expression)
`kbase` (module_expression)
`kbase` (ideal_expression`,` int_expression)
`kbase` (module_expression`,` int_expression)

Type: the same as the input type of the first argument

Purpose: With one argument: computes a vector space basis (consisting of monomials) of the quotient ring by the ideal, resp. of a free module by the module, in case it is finite dimensional and if the input is a standard basis with respect to the ring ordering.
Note that, if the input is not a standard basis, the leading terms of the input are used and the result may have no meaning.
With two arguments: computes the part of a vector space basis of the respective quotient with degree of the monomials equal to the second argument. Here, the quotient

does not need to be finite dimensional. If an attribute`isHomog` (of type `intvec`) is present, it is used as module weight.

Example:

```
  ring r=32003,(x,y,z),ds;
  ideal i=x2,y2,z;
  kbase(std(i));
↦ _[1]=xy
↦ _[2]=y
↦ _[3]=x
↦ _[4]=1
  i=x2,y3,xyz;  // quotient not finite dimensional
  kbase(std(i),2);
↦ _[1]=z2
↦ _[2]=yz
↦ _[3]=xz
↦ _[4]=y2
↦ _[5]=xy
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.168 [vdim], page 286.

5.1.70 kernel

Syntax: `kernel (` ring_name, map_name `)`

Type: ideal

Purpose: returns the kernel of a given map.
The second argument has to be a map from the basering to the given ring (or an ideal defining such a map).

Example:

```
  ring r1=32003,(x,y,z,w),lp;
  ring r=32003,(x,y,z),dp;
  ideal i=x,y,z;
  map f=r1,i;
  setring r1;
  // the kernel of f
  kernel(r,f);
↦ _[1]=w
```

See Section D.4.2.5 [alg_kernel], page 1007; Section D.4.11.15 [hom_kernel], page 1109; Section 4.6 [ideal], page 80; Section 4.12 [map], page 106; Section 5.1.94 [modulo], page 223; Section 5.1.117 [preimage], page 240; Section 4.20 [ring], page 127.

5.1.71 kill

Syntax: `kill` name
`kill` list_of_names

Type: none

Purpose: deletes objects.

Example:

```
  int i=3;
  ring r=0,x,dp;
  poly p;
  listvar();
↦ // r                              [0]  *ring
↦ //      p                              [0]  poly
↦ // i                              [0]  int 3
  kill i,r;
  // the variable ‘i‘ does not exist any more
  i;
↦    ? ‘i‘ is undefined
↦    ? error occurred in or before ./examples/kill.sing line 7: ‘  i;‘
  listvar();
```

See Section 5.1.18 [defined], page 170; Section D.2.3 [general_lib], page 805.

5.1.72 killattrib

Syntax: `killattrib` (name)
`killattrib` (name, string_expression)

Type: none

Purpose: deletes all attributes respective the attribute given as the second argument.

Example:

```
  ring r=32003,(x,y),lp;
  ideal i=maxideal(1);
  attrib(i,"isSB",1);
  attrib(i);
↦ attr:isSB, type int
  killattrib(i,"isSB");
  attrib(i);
↦ no attributes
  attrib(i,"isSB",1);
  killattrib(i);
  attrib(i);
↦ no attributes
```

See Section 5.1.2 [attrib], page 156; Section 5.1.111 [option], page 234.

5.1.73 koszul

Syntax: `koszul` (int_expression, int_expression)
`koszul` (int_expression, ideal_expression)
`koszul` (int_expression, int_expression, ideal_expression)

Type: matrix

Purpose: `koszul(d,n)` computes a matrix of the Koszul relations of degree d of the first n ring variables.

`koszul(d,id)` computes a matrix of the Koszul relations of degree d of the generators of the ideal `id`.

`koszul(d,n,id)` computes a matrix of the Koszul relations of degree d of the first n generators of the ideal `id`.

Note: koszul(1,id), koszul(2,id), ... form a complex, that is, the product of the matrices koszul(i,id) and koszul(i+1,id) equals zero.

Example:

```
  ring r=32003,(x,y,z),dp;
  print(koszul(2,3));
↦ -y,-z,0,
↦ x, 0, -z,
↦ 0, x, y
  ideal I=xz2+yz2+z3,xyz+y2z+yz2,xy2+y3+y2z;
  print(koszul(1,I));
↦ xz2+yz2+z3,xyz+y2z+yz2,xy2+y3+y2z
  print(koszul(2,I));
↦ -xyz-y2z-yz2,-xy2-y3-y2z,0,
↦ xz2+yz2+z3,  0,          -xy2-y3-y2z,
↦ 0,           xz2+yz2+z3, xyz+y2z+yz2
  print(koszul(2,I)*koszul(3,I));
↦ 0,
↦ 0,
↦ 0
```

See Section 4.7 [int], page 85; Section 4.13 [matrix], page 108.

5.1.74 laguerre

Syntax: laguerre (poly_expression)
laguerre (poly_expression , int_expression , int_expression)

Type: list

Purpose: In characteristic 0:
computes all complex roots of a univariate polynomial using Laguerre's algorithm. The second argument defines the precision of the fractional part if the ground field is the field of rational numbers, otherwise it will be ignored (default: 10). third argument (can be 0, 1 or 2) gives the number of extra runs for Laguerre's algorithm (with corrupted roots), leading to better results (default: 1).
In characteristic p:
computes all roots of a univariate polynomial using factorization

Note: If the ground field is the field of complex numbers, the elements of the list are of type number, otherwise of type string.

Example:

```
ring rs1=0,(x,y),lp;
poly f=15x5+x3+x2-10;
laguerre(f);
↦ [1]:
↦    0.8924637479
↦ [2]:
↦    (-0.7392783383+I*0.5355190078)
↦ [3]:
↦    (-0.7392783383-I*0.5355190078)
↦ [4]:
↦    (0.2930464644-I*0.9003002396)
↦ [5]:
```

```
↦    (0.2930464644+I*0.9003002396)
laguerre(f,30,2);
↦ [1]:
↦    0.892463747926447366650738925573
↦ [2]:
↦    (-0.739278338347345779837700592458+I*0.535519007804239739420495882967
↦ [3]:
↦    (-0.739278338347345779837700592458-I*0.535519007804239739420495882967
↦ [4]:
↦    (0.293046464384122096512331129672-I*0.90030023962350314542842479521)
↦ [5]:
↦    (0.293046464384122096512331129672+I*0.90030023962350314542842479521)
```

5.1.75 lead

Syntax: `lead` (poly_expression)
`lead` (vector_expression)
`lead` (ideal_expression)
`lead` (module_expression)

Type: the same as the input type

Purpose: returns the leading (or initial) term(s) of a polynomial, a vector, resp. of the generators of an ideal or module with respect to the monomial ordering.

Note: `IN` may be used instead of `lead`.

Example:

```
  ring r=32003,(x,y,z),(c,ds);
  poly f=2x2+3y+4z3;
  vector v=[2x10,f];
  ideal i=f,z;
  module m=v,[0,0,2+x];
  lead(f);
↦ 3y
  lead(v);
↦ [2x10]
  lead(i);
↦ _[1]=3y
↦ _[2]=z
  lead(m);
↦ _[1]=[2x10]
↦ _[2]=[0,0,2]
  lead(0);
↦ 0
```

See Section 4.6 [ideal], page 80; Section 5.1.76 [leadcoef], page 209; Section 5.1.77 [leadexp], page 210; Section 5.1.78 [leadmonom], page 210; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.76 leadcoef

Syntax: `leadcoef` (poly_expression)
`leadcoef` (vector_expression)

Type: number

Purpose: returns the leading (or initial) coefficient of a polynomial or a vector with respect to the monomial ordering.

Example:

```
  ring r=32003,(x,y,z),(c,ds);
  poly f=x2+y+z3;
  vector v=[2*x^10,f];
  leadcoef(f);
↦ 1
  leadcoef(v);
↦ 2
  leadcoef(0);
↦ 0
```

See Section 5.1.75 [lead], page 209; Section 5.1.77 [leadexp], page 210; Section 5.1.78 [leadmonom], page 210; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.77 leadexp

Syntax: `leadexp` (poly_expression)
`leadexp` (vector_expression)

Type: intvec

Purpose: returns the exponent vector of the leading monomial of a polynomial or a vector. In the case of a vector the last component is the index in the vector. (The inverse to `monomial`.)

Example:

```
  ring r=32003,(x,y,z),(c,ds);
  poly f=x2+y+z3;
  vector v=[2*x^10,f];
  leadexp(f);
↦ 0,1,0
  leadexp(v);
↦ 10,0,0,1
  leadexp(0);
↦ 0,0,0
```

See Section 4.9 [intvec], page 93; Section 5.1.75 [lead], page 209; Section 5.1.76 [leadcoef], page 209; Section 5.1.78 [leadmonom], page 210; Section 5.1.96 [monomial], page 225; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.78 leadmonom

Syntax: `leadmonom` (poly_expression)
`leadmonom` (vector_expression)

Type: the same as the input type

Purpose: returns the leading monomial of a polynomial or a vector as a polynomial or vector whose coefficient is one.

Example:

```
  ring r=32003,(x,y,z),(c,ds);
  poly f=2x2+3y+4z3;
  vector v=[0,2x10,f];
  leadmonom(f);
↦ y
  leadmonom(v);
↦ [0,x10]
  leadmonom(0);
↦ 0
```

See Section 4.9 [intvec], page 93; Section 5.1.75 [lead], page 209; Section 5.1.76 [leadcoef], page 209; Section 5.1.77 [leadexp], page 210; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.79 LIB

Syntax: `LIB` string_expression`;`

Type: none

Purpose: reads a library of procedures from a file. In contrast to the command `load`, the procedures from the library are added to the package `Top` as well as the package corresponding to the library. If the given filename does not start with `.` or `/` and cannot be located in the current directory, each directory contained in the library `SearchPath` is searched for file of this name. See Section 3.8.11 [Loading a library], page 66, for more info on `SearchPath`.

Note on standard.lib:
Unless Singular is started with the `--no-stdlib` option, the library `standard.lib` is automatically loaded at start-up time.

Example:

```
  option(loadLib); // show loading of libraries

                   // the names of the procedures of inout.lib
  LIB "inout.lib"; // are now known to Singular
↦ // ** loaded inout.lib (4.1.2.0,Feb_2019)
```

See Section 3.1.6 [Command line options], page 19; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 793; Section 5.2.12 [load], page 299; Section 4.16 [package], page 119; Section 4.18 [proc], page 124; Section D.1 [standard_lib], page 793; Section 4.22 [string], page 130; Section 5.1.155 [system], page 275.

5.1.80 lift

Syntax: `lift (` ideal_expression`,` subideal_expression `)`
`lift (` module_expression`,` submodule_expression `)`
`lift (` ideal_expression`,` subideal_expression`,` matrix_name `)`
`lift (` module_expression`,` submodule_expression`,` matrix_name `)`
`lift (` ideal_expression`,` subideal_expression`,` matrix_name`,` string_expression `)`
`lift (` module_expression`,` submodule_expression`,` matrix_name`,` string_expression `)`

Type: matrix

Purpose: computes the transformation matrix which expresses the generators of a submodule in terms of the generators of a module. Depending on which algorithm is used, modules

are represented by a standard basis, or not.
More precisely, if `m` is the module (or ideal), `sm` the submodule (or ideal), and `T` the transformation matrix returned by lift, then `matrix(sm)*U = matrix(m)*T` and `module(sm*U) = module(matrix(m)*T)` (resp. `ideal(sm) = ideal(matrix(m)*T)`), where `U` is a diagonal matrix of units.
`U` is always the identity if the basering is a polynomial ring (not power series ring). `U` is stored in the optional third argument.

Note: Gives a warning if `sm` is not a submodule.
An optional 4th argument specifies the Groebner base algorithm to use. Possible values are `"std"` and `"slimgb"`.

Example:

```
      ring r=32003,(x,y,z),(dp,C);
      ideal m=3x2+yz,7y6+2x2y+5xz;
      poly f=y7+x3+xyz+z2;
      ideal i=jacob(f);
      matrix T=lift(i,m);
      matrix(m)-matrix(i)*T;
    ↦ _[1,1]=0
    ↦ _[1,2]=0
```

See Section 5.1.26 [division], page 174; Section 4.6 [ideal], page 80; Section 5.1.81 [liftstd], page 212; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271; Section 5.1.156 [syz], page 280.

5.1.81 liftstd

Syntax: `liftstd (` ideal_expression`,` matrix_name`[,` module_name`] [,` string_expression `] [,` ideal_expression `])`
`liftstd (` module_expression`,` matrix_name`[,` module_name`] [,` string_expression `] [,` module_expression `])`

Type: ideal or module

Purpose: returns a standard basis of an ideal or module and the transformation matrix from the given ideal, resp. module, to the standard basis.
That is, if `m` is the ideal or module, `sm` the standard basis returned by `liftstd`, and `T` the transformation matrix (`sm=liftstd(m,T)`) then `matrix(sm)=matrix(m)*T` and `sm=ideal(matrix(m)*T)`, resp. `sm=module(matrix(m)*T)`. If working in a quotient ring, then `matrix(sm)=reduce(matrix(m)*T,0)` and `sm=reduce(ideal(matrix(m)*T),0)`.
If a module name is given as a third argument, the syzygy module will be returned. (`sm=liftstd(m,T,s)` then additional `matrix(m)*matrix(s)=0`).
An optional string argument specifies the Groebner base algorithm to use. Possible values are `"std"` and `"slimgb"`.
Given an optional last argument (say `n`), the algorithm computes a standard bases of `(m+n)`, syzygies of `m` modulo `n`, and the transformation matrix only for `m`. These are relative transformation matrix resp. the syzygy module of `n` modulo `m`. (For syzygies, the same can be achieved using Section 5.1.94 [modulo], page 223.)

Example:

```
      ring R=0,(x,y,z),dp;
      poly f=x3+y7+z2+xyz;
      ideal i=jacob(f);
```

```
  matrix T;
  ideal sm=liftstd(i,T);
  sm;
↦ sm[1]=xy+2z
↦ sm[2]=3x2+yz
↦ sm[3]=yz2+3048192z3
↦ sm[4]=3024xz2-yz2
↦ sm[5]=y2z-6xz
↦ sm[6]=3097158156288z4+2016z3
↦ sm[7]=7y6+xz
  print(T);
↦ 0,1,T[1,3],   T[1,4],y,  T[1,6],0,
↦ 0,0,-3x+3024z,3x,    0,  T[2,6],1,
↦ 1,0,T[3,3],   T[3,4],-3x,T[3,6],0
  matrix(sm)-matrix(i)*T;
↦ _[1,1]=0
↦ _[1,2]=0
↦ _[1,3]=0
↦ _[1,4]=0
↦ _[1,5]=0
↦ _[1,6]=0
↦ _[1,7]=0
  module s;
  sm=liftstd(i,T,s);
  print(s);
↦ -xy-2z,0,     s[1,3],s[1,4],xyz+2z2,  -14y5z+x2z,
↦ 0,     -xy-2z,s[2,3],s[2,4],-3x2y-6xz,-3x3+2z2,
↦ 3x2+yz,7y6+xz,7y6+xz,s[3,4],21xy6-yz2,21x2y5-xz2
  matrix(i)*matrix(s);
↦ _[1,1]=0
↦ _[1,2]=0
↦ _[1,3]=0
↦ _[1,4]=0
↦ _[1,5]=0
↦ _[1,6]=0
```

See Section 5.1.26 [division], page 174; Section 4.6 [ideal], page 80; Section 5.1.80 [lift], page 211; Section 4.13 [matrix], page 108; Section 5.1.94 [modulo], page 223; Section 5.1.111 [option], page 234; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271; Section 5.1.156 [syz], page 280.

5.1.82 listvar

Syntax: `listvar` ([package])
`listvar` ([package,] type)
`listvar` ([package,] ring_name)
`listvar` ([package,] name)
`listvar` ([package,] `all`)

Type: none

Purpose: lists all (user-)defined names:

- `listvar()`: all currently visible names except procedures in the current namespace,
- `listvar`(type): all currently visible names of the given type,

- `listvar(`ring_name`)`: all names which belong to the given ring,
- `listvar(`name`)`: the object with the given name,
- `listvar(all)`: all names except procedures in the current and `Top` namespace.

The current basering is marked with a `*`. The nesting level of variables in procedures is shown in square brackets.

package can be `Current`, `Top` or any other identifier of type package.

Example:

```
  proc t1 { }
  proc t2 { }
  ring s;
  poly ss;
  ring r;
  poly f=x+y+z;
  int i=7;
  ideal I=f,x,y;
  listvar();
↦ // i                          [0]  int 7
↦ // r                          [0]  *ring
↦ //      I                          [0]  ideal, 3 generator(s)
↦ //      f                          [0]  poly
↦ // s                          [0]  ring
  listvar(r);
↦ // r                          [0]  *ring
↦ // I                          [0]  ideal, 3 generator(s)
↦ // f                          [0]  poly
  listvar(t1);
↦ // t1                         [0]  proc
  listvar(proc);
↦ // t2                         [0]  proc
↦ // t1                         [0]  proc
↦ // mathicgb_prOrder           [0]  proc from singmathic.so (C)
↦ // mathicgb                   [0]  proc from singmathic.so (C)
↦ // create_ring                [0]  proc from standard.lib
↦ // min                        [0]  proc from standard.lib
↦ // max                        [0]  proc from standard.lib
↦ // datetime                   [0]  proc from standard.lib
↦ // weightKB                   [0]  proc from standard.lib
↦ // fprintf                    [0]  proc from standard.lib
↦ // printf                     [0]  proc from standard.lib
↦ // sprintf                    [0]  proc from standard.lib
↦ // quotient4                  [0]  proc from standard.lib
↦ // quotient5                  [0]  proc from standard.lib
↦ // quotient3                  [0]  proc from standard.lib
↦ // quotient2                  [0]  proc from standard.lib
↦ // quotient1                  [0]  proc from standard.lib
↦ // quot                       [0]  proc from standard.lib
↦ // res                        [0]  proc from standard.lib
↦ // groebner                   [0]  proc from standard.lib
↦ // qslimgb                    [0]  proc from standard.lib
↦ // hilbRing                   [0]  proc from standard.lib
↦ // par2varRing                [0]  proc from standard.lib
↦ // quotientList                    [0]  proc from standard.lib
↦ // stdhilb                         [0]  proc from standard.lib
↦ // stdfglm                         [0]  proc from standard.lib
↦ // Float                           [0]  proc from kernel (C)
↦ // crossprod                       [0]  proc from kernel (C)
  LIB "polylib.lib";
  listvar(Poly);
↦    ? Poly is undefined
↦    ? error occurred in or before ./examples/listvar.sing line 14: ‘  lis
  var(Poly);‘
```

See Section 3.5.3 [Names], page 44; Section 3.7.4 [Names in procedures], page 54; Section 5.1.18 [defined], page 170; Section 5.1.103 [names], page 229; Section 4.16 [package], page 119; Section 5.1.160 [type], page 282.

5.1.83 lres

Syntax: `lres` (ideal_expression`,` int_expression)

Type: resolution

Purpose: computes a free resolution of an ideal using LaScala's algorithm.

More precisely, let R be the basering and I be the given ideal. Then `lres` computes a minimal free resolution of R/I

$$\ldots \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} R \longrightarrow R/I \longrightarrow 0.$$

If the int_expression k is not zero then the computation stops after k steps and returns a list of modules $M_i = \texttt{module}(A_i)$, i=1..k.

`list L=lres(I,0);` returns a list L of n modules (where n is the number of variables of the basering) such that `L[i]` $= M_i$ in the above notation.

Note: The ideal_expression has to be homogeneous.

Accessing single elements of a resolution may require that some partial computations have to be finished and may therefore take some time.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=xz,yz,x3-y3;
  def L=lres(I,0);
  print(betti(L),"betti");
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     2     1
↦     2:     -     1     1
↦ ------------------------
↦ total:     1     3     2
↦
  L[2];     // the first syzygy module of r/I
↦ _[1]=-x*gen(1)+y*gen(2)
↦ _[2]=-x2*gen(2)+y2*gen(1)+z*gen(3)
```

See Section 5.1.4 [betti], page 159; Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 4.7 [int], page 85; Section 5.1.93 [minres], page 223; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.99 [mres_map], page 226; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.84 ludecomp

qcindex Gauss

Syntax: `ludecomp` (matrix_expression)

Type: list

Purpose: Computes the LU-decomposition of an (m x n) matrix.

The matrix, A say, must consist of numbers, only. This means that when the basering represents some $K[x_1, x_2, \ldots, x_r]$, then all entries of A must come from the ground field K.
The LU-decomposition of A is a triple of matrices P, L, and U such that
- P * A = L * U,
- P is an (m x m) permutation matrix, i.e., its rows/columns form the standard basis of K^m,
- L is an (m x m) matrix in lower triangular form with all diagonal entries equal to 1, and
- U is an (m x n) matrix in upper row echelon form.
From these conditions, it easily follows that also A = P * L * U holds, since P is self-inverse.

`list L=ludecomp(A);` fills a list L with the three above entries P, L, and U.

Example:

```
  ring r=0,(x),dp;
  matrix A[3][4]=1,2,3,4,1,1,1,1,2,2,1,1;
  list plu = ludecomp(A);
  print(plu[3]);                    // the matrix U of the decomposition
↦ 1,2, 3, 4,
↦ 0,-1,-2,-3,
↦ 0,0, -1,-1
  print(plu[1]*A-plu[2]*plu[3]);   // should be the zero matrix
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
```

See `system("rref",A);`. See Section 5.1.85 [luinverse], page 216; Section 5.1.86 [lusolve], page 217; Section 5.1.155 [system], page 275.

5.1.85 luinverse

qcindex Gauss

Syntax: `luinverse` (matrix_expression)

Type: matrix

Syntax: `luinverse` (matrix_expression, matrix_expression, matrix_expression)

Type: matrix

Purpose: Computes the inverse of a matrix A, if A is invertible.

The matrix A must be given either directly, or by its LU-decomposition. In the latter case, three matrices P, L, and U are expected, in this order, which satisfy
- P * A = L * U,
- P is an (m x m) permutation matrix, i.e., its rows/columns form the standard basis

of K^m,
- L is an (m x m) matrix in lower triangular form with all diagonal entries equal to 1, and
- U is an (m x m) matrix in upper row echelon form.
Then, the inverse of A exists if and only if U is invertible, and one has $A^{-1} = U^{-1} \cdot L^{-1} \cdot P$, since P is self-inverse.
In the case of A being given directly, `luinverse` first computes its LU-decomposition, and then proceeds as in the case when P, L, and U are provided.

`list L=luinverse(A);` fills the list L with either one entry = 0 (signaling that A is not invertible), or with the two entries $1, A^{-1}$. Thus, in either case the user may first check the condition `L[1]==1` to find out whether A is invertible.

Note: The method will give a warning for any non-quadratic matrix A.

Example:

```
  ring r=0,(x),dp;
  matrix A[3][3]=1,2,3,1,1,1,2,2,1;
  list L = luinverse(A);
  if (L[1] == 1)
  {
    print(L[2]);
    "----- next should be the (3 x 3)-unit matrix:";
    print(A*L[2]);
  }
↦ -1,4, -1,
↦ 1, -5,2,
↦ 0, 2, -1
↦ ----- next should be the (3 x 3)-unit matrix:
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
```

See `system("rref",A);`. See Section 5.1.84 [ludecomp], page 216; Section 5.1.86 [lusolve], page 217; Section 5.1.155 [system], page 275.

5.1.86 lusolve

qcindex Gauss

Syntax: `lusolve (` matrix_expression`,` matrix_expression`,` matrix_expression`,` matrix_expression `)`

Type: matrix

Purpose: Computes all solutions of a linear equation system A*x = b, if solvable
The (m x n matrix A must be given by its LU-decomposition, that is, by three matrices P, L, and U, in this order, which satisfy
- P * A = L * U,
- P is an (m x m) permutation matrix, i.e., its rows/columns form the standard basis of K^m,
- L is an (m x m) matrix in lower triangular form with all diagonal entries equal to 1, and
- U is an (m x n) matrix in upper row echelon form.
The fourth argument, b, is expected to be an (m x 1) matrix.

`list Q=lusolve(P,L,U,b);` fills the list Q with either one entry = 0 (signaling that A*x=b has no solution), or with the three entries 1, x, H, where x is any (n x 1) solution of the given linear system, and H is a matrix the columns of which span the solution space of the homogeneous linear system. (I.e., `ncols(H)` is the dimension of the solution space.)
If there is exactly one solution, then H is the 1x1 matrix with entry zero.

Note: The method will give a warning if the matrices violate the above conditions regarding row and column numbers, or if the number of rows of the vector b does not equal m. The method expects matrices with entries coming from the ground field of the given polynomial ring, only.

Example:

```
  ring r=0,(x),dp;
  matrix A[4][4]=1,1,1,0,1,2,3,1,1,3,5,2,1,4,7,3;
  matrix b[4][1]=2,5,8,11;
  list L=ludecomp(A);
  list Q=lusolve(L[1],L[2],L[3],b);
  if (Q[1] == 1)
  {
    "one solution:";
    print(Q[2]);
    "check whether result is correct (iff next is zero vector):";
    print(A*Q[2]-b);
    if ((nrows(Q[3])==1) and (ncols(Q[3])==1) and (Q[3][1,1]==0))
    { "printed solution is the only solution to given linear system" }
    else
    {
      "homogeneous solution space is spanned by columns of:";
      print(Q[3]);
    }
  }
↦ one solution:
↦ -1,
↦ 3,
↦ 0,
↦ 0
↦ check whether result is correct (iff next is zero vector):
↦ 0,
↦ 0,
↦ 0,
↦ 0
↦ homogeneous solution space is spanned by columns of:
↦ -1,-1,
↦ 1, 2,
↦ 0, -1,
↦ -1,0
```

See `system("rref",A);`. See Section 5.1.84 [ludecomp], page 216; Section 5.1.85 [luinverse], page 216; Section 5.1.155 [system], page 275.

5.1.87 max

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: max (i_1, ..., i_k)

Type: same as type of i_1, ..., i_k resp.

Purpose: returns the maximum for any arguments of a type
for which '>' is defined

Example:

```
  // biggest int
max(2,3);
↦ 3
max(1,4,3);
↦ 4
// lexicographically biggest intvec
max(intvec(1,2),intvec(0,1),intvec(1,1));
↦ 1,2
// polynopmial with biggest leading monomial
ring r = 0,x,dp;
max(x+1,x2+x);
↦ x2+x
```

See also: Section 5.1.90 [min], page 220.

5.1.88 maxideal

Syntax: `maxideal` (int_expression)

Type: ideal

Purpose: returns the power given by int_expression of the maximal ideal generated by all ring variables (`maxideal(i)=1` for `i<=0`).

Example:

```
  ring r=32003,(x,y,z),dp;
  maxideal(2);
↦ _[1]=z2
↦ _[2]=yz
↦ _[3]=y2
↦ _[4]=xz
↦ _[5]=xy
↦ _[6]=x2
```

See Section 4.6 [ideal], page 80; Section 4.20 [ring], page 127.

5.1.89 memory

Syntax: `memory` (int_expression)

Type: bigint

Purpose: returns statistics concerning the memory management:

- `memory(0)` is the number of active (used) bytes,
- `memory(1)` is the number of bytes allocated from the operating system,
- `memory(2)` is the maximal number of bytes ever allocated from the operating system during the current Singular session.

Note: To monitor the memory usage during ongoing computations the option `mem` should be set (using the command `option(mem);`, see also Section 5.1.111 [option], page 234).

Example:

```
  ring r=0,(x(1..500)),dp;
  poly p=(x(1)+x(500))^50;
  proc ReportMemoryUsage()
  {  "Memory currently used by SINGULAR     :",memory(0),"Byte (",
     int(memory(0) div 1024), "KByte)" +newline+
     "Memory currently allocated from system:",memory(1), "Byte (",
     int(memory(1) div 1024), "KByte)";
     "Maximal memory allocated from system  :",memory(2), "Byte (",
     int(memory(2) div 1024), "KByte)";
  }
  ReportMemoryUsage();
↦ Memory currently used by SINGULAR     : 154560 Byte ( 151 KByte)
↦ Memory currently allocated from system: 2232320 Byte ( 2180 KByte)
↦ Maximal memory allocated from system  : 2232320 Byte ( 2180 KByte)
  kill p;
  ReportMemoryUsage(); // less memory used: p killed
↦ Memory currently used by SINGULAR     : 83992 Byte ( 82 KByte)
↦ Memory currently allocated from system: 2232320 Byte ( 2180 KByte)
↦ Maximal memory allocated from system  : 2232320 Byte ( 2180 KByte)
  kill r;
  ReportMemoryUsage(); // even less memory: r killed
↦ Memory currently used by SINGULAR     : 71752 Byte ( 70 KByte)
↦ Memory currently allocated from system: 2232320 Byte ( 2180 KByte)
↦ Maximal memory allocated from system  : 2232320 Byte ( 2180 KByte)
```

See Section 5.1.111 [option], page 234; Section 5.1.155 [system], page 275.

5.1.90 min

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: max (i_1, ..., i_k)

Type: same as type of i_1, ..., i_k resp.

Purpose: returns the maximum for any arguments of a type
for which '>' is defined

Example:

```
  // biggest int
max(2,3);
↦ 3
max(1,4,3);
↦ 4
// lexicographically biggest intvec
max(intvec(1,2),intvec(0,1),intvec(1,1));
↦ 1,2
// polynopmial with biggest leading monomial
ring r = 0,x,dp;
max(x+1,x2+x);
↦ x2+x
```

See also: Section 5.1.90 [min], page 220.

5.1.91 minbase

Syntax: `minbase` (ideal_expression)
`minbase` (module_expression)

Type: the same as the type of the argument

Purpose: returns a minimal set of generators of an ideal, resp. module, if the input is either homogeneous or if the ordering is local.

Note: this command is not available over coefficient rings.

Example:

```
  ring r=181,(x,y,z),(c,ls);
  ideal id2=x2+xyz,y2-z3y,z3+y5xz;
  ideal id4=maxideal(3)+id2;
  size(id4);
↦ 13
  minbase(id4);
↦ _[1]=x2
↦ _[2]=xyz+x2
↦ _[3]=xz2
↦ _[4]=y2
↦ _[5]=yz2
↦ _[6]=z3
```

See Section 5.1.100 [mstd], page 227.

5.1.92 minor

Syntax: `minor` (matrix_expression M, int_expression mSize,
[ideal_expression I],
[int_expression k],
[string_expression algorithm],
[int_expression cachedP],
[int_expression cachedM])

Type: ideal

Purpose: returns the specified set of (mSize x mSize)-minors (= subdeterminants) of the given matrix M. These minors form the list of generators of the returned ideal.
If the optional ideal I is given, it is assumed to capture a standard basis. In this case, all computations will be performed modulo I.
If k is not given, all minors will be computed. Otherwise, if k > 0, the first k non-zero minors will be computed; for k < 0, the first |k| minors will be computed regardless whether they are zero or not. Here, "first k minors" is with respect to a fixed ordering among all minors. (To understand the ordering, run the below example, type `minor(m,2,i,18);` and inspect the ordering among the returned 18 minors. Note that this ordering is only enforced when some k <> 0 is provided. Otherwise, no ordering among the returned minors can be guaranteed. This is due to the fact that in this case, `minor` may call a specially tuned implementation of Bareiss's algorithm.)
If no algorithm is given, a heuristic will pick the best-suited algorithm among Bareiss's algorithm (which is only applicable over integral domains), Laplace's algorithm, and Laplace's algorithm combined with caching of subdeterminantes. In the heuristic setting, cacheP and cacheM must also be absent.

If the argument algorithm is present it must be one of `B/bareiss`, `L/laplace`, and `C/cache`. For, `B/bareiss` and `L/laplace` the optional arguments cacheP and cacheM must again be absent, whereas for `C/cache`, they may be provided: cachedP determines the maximum number of cached subdeterminantes (=polynomials), and cachedM the total number of cached monomials (counted over all cached polynomials). If, for algorithm = `C/cache` cachedP and cachedM are not provided by the user, the values 200 and 100000, respectively, will be used as defaults.

Note: If `mSize` is larger than the given matrix, `minor` returns 0, if `mSize` is smaller than 1, `minor` returns 1.

Example:

```
  ring r=0,(a,b,c,d,e,f,g,h,s,t,u,v),ds;
  matrix m[3][4]=a,b,c,d,e,f,g,h,s,t,u,v;
  print(m);
↦ a,b,c,d,
↦ e,f,g,h,
↦ s,t,u,v
  // let’s compute all non-zero minors;
  // here we do not guarantee any ordering:
  minor(m,2);
↦ _[1]=-hu+gv
↦ _[2]=-ht+fv
↦ _[3]=-hs+ev
↦ _[4]=-du+cv
↦ _[5]=-dt+bv
↦ _[6]=-ds+av
↦ _[7]=gt-fu
↦ _[8]=gs-eu
↦ _[9]=ct-bu
↦ _[10]=cs-au
↦ _[11]=-fs+et
↦ _[12]=-bs+at
↦ _[13]=-dg+ch
↦ _[14]=-df+bh
↦ _[15]=-de+ah
↦ _[16]=cf-bg
↦ _[17]=ce-ag
↦ _[18]=-be+af
  ideal i=a,c; i=std(i);
  // here come the first 4 non-zero minors mod I;
  // this time, a fixed ordering is guaranteed:
  minor(m,2,i,4);
↦ _[1]=-be
↦ _[2]=bg
↦ _[3]=-de
↦ _[4]=-df+bh
  // and here the first 4 minors mod I (possibly zero)
  // using Laplace’s algorithm,
  // again, the fixed ordering is guaranteed:
  minor(m,2,i,-4,"Laplace");
↦ _[1]=-be
↦ _[2]=0
↦ _[3]=bg
↦ _[4]=-de
```

See Section 5.1.23 [det], page 172.

5.1.93 minres

Syntax: `minres` (list_expression)

Type: list

Syntax: `minres` (resolution_expression)

Type: resolution

Purpose: minimizes a free resolution of an ideal or module given by the list_expression, resp. resolution_expression.

Example:

```
  ring r1=32003,(x,y),dp;
  ideal i=x5+xy4,x3+x2y+xy2+y3;
  resolution rs=lres(i,0);
  rs;
↦   1       2       1
↦ r1 <--  r1 <--  r1
↦ 
↦ 0       1       2
↦ 
  list(rs);
↦ [1]:
↦    _[1]=x3+x2y+xy2+y3
↦    _[2]=xy4
↦ [2]:
↦    _[1]=xy4*gen(1)-x3*gen(2)-x2y*gen(2)-xy2*gen(2)-y3*gen(2)
  minres(rs);
↦   1       2       1
↦ r1 <--  r1 <--  r1
↦ 
↦ 0       1       2
↦ 
  list(rs);
↦ [1]:
↦    _[1]=x3+x2y+xy2+y3
↦    _[2]=xy4
↦ [2]:
↦    _[1]=xy4*gen(1)-x3*gen(2)-x2y*gen(2)-xy2*gen(2)-y3*gen(2)
```

See Section 5.1.48 [fres], page 188; Section 5.1.98 [mres], page 225; Section 5.1.99 [mres_map], page 226; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.94 modulo

Syntax: `modulo` (ideal_expression, ideal_expression)
`modulo` (module_expression, module_expression)
`modulo` (ideal_expression, ideal_expression, string_expression)
`modulo` (module_expression, module_expression, string_expression)
`modulo` (ideal_expression, ideal_expression, matrix_name)

Type: module

Purpose: `modulo(h1,h2)` represents $h_1/(h_1 \cap h_2) \cong (h_1+h_2)/h_2$ where h_1 and h_2 are considered as submodules of the same free module R^l (l=1 for ideals). Let H_1, resp. H_2, be the matrices of size $l \times k$, resp. $l \times m$, having the generators of h_1, resp. h_2, as columns. Then $h_1/(h_1 \cap h_2) \cong R^k/ker(\overline{H_1})$ where $\overline{H_1} : R^k \to R^l/Im(H_2) = R^l/h_2$ is the induced map.
`modulo(h1,h2)` returns generators of the kernel of this induced map.
An optional third string argument give the GB algorithm to use. (default: "std", else: "groebner", "modstd", "slimgb").
An optional third argument (a name of a matrix) gives the matrix to store the transformation of the input to the GB (see Section 5.1.81 [liftstd], page 212).

Note: If for at least one of `h1` or `h2` the attribute `"isHomog"` is set, `modulo(h1,h2)` also sets the attribute `"isHomog"` (if possible, that is, if the weights are compatible).

Example:

```
  ring r;
  ideal h1=x,y,z;
  ideal h2=x;
  module m=modulo(h1,h2);
  print(m);
↦ 1,0, 0,0,
↦ 0,-z,x,0,
↦ 0,y, 0,x
```

See Section D.4.11.15 [hom_kernel], page 1109; Section 5.1.156 [syz], page 280.

5.1.95 monitor

Syntax: `monitor` (link_expression)
`monitor` (link_expression, string_expression)

Type: none

Purpose: controls the recording of all user input and/or program output into a file. The second argument describes what to log: `"i"` means input, `"o"` means output, `"io"` for both. The default for the second argument is `"i"`.
Each `monitor` command closes a previous monitor file and opens the file given by the first string expression.
`monitor ("")` turns off recording.

Example:

```
  monitor("doe.tmp","io"); // log input and output to doe.tmp
  ring r;
  poly f=x+y+z;
  int i=7;
  ideal I=f,x,y;
  monitor("");             // stop logging:
  // doe.tmp contains now all input and output from the example above
```

See Section 4.10.2 [link expressions], page 97.

5.1.96 monomial

Syntax: `monomial` (intvec_expression)

Type: poly resp. vector

Purpose: converts an integer vector to a power product (the inverse to `leadexp`). Returns a `vector` iff the length of the argument is number of variables +1.

Example:

```
  ring r=0,(x,y,z),dp;
  monomial(intvec(2,3));
↦ x2y3
  monomial(intvec(2,3,0,1));
↦ x2y3*gen(1)
  leadexp(monomial(intvec(2,3,0,1)));
↦ 2,3,0,1
```

See Section 4.9 [intvec], page 93; Section 5.1.77 [leadexp], page 210.

5.1.97 mpresmat

Syntax: `mpresmat` (ideal_expression, int_expression)

Type: module

Purpose: computes the multipolynomial resultant matrix of the input system. Uses the sparse resultant matrix method of Gelfand, Kapranov and Zelevinsky (second parameter = 0) or the resultant matrix method of Macaulay (second parameter = 1).

Note: When using the resultant matrix method of Macaulay the input system must be homogeneous. The number of elements in the input system must be the number of variables in the basering plus one.

Example:

```
  ring rsq=(0,s,t,u),(x,y),lp;
  ideal i=s+tx+uy,x2+y2-10,x2+xy+2y2-16;
  module m=mpresmat(i,0);
  print(m);
↦ -16,0,  -10,0,  (s),0,  0,  0,  0,  0,
↦ 0,  -16,0,  -10,(u),(s),0,  0,  0,  0,
↦ 2,  0,  1,  0,  0,  (u),0,  0,  0,  0,
↦ 0,  2,  0,  1,  0,  0,  0,  0,  0,  0,
↦ 0,  0,  0,  0,  (t),0,  -10,(s),0,  -16,
↦ 1,  0,  0,  0,  0,  (t),0,  (u),(s),0,
↦ 0,  1,  0,  0,  0,  0,  1,  0,  (u),2,
↦ 1,  0,  1,  0,  0,  0,  0,  (t),0,  0,
↦ 0,  1,  0,  1,  0,  0,  0,  0,  (t),1,
↦ 0,  0,  0,  0,  0,  0,  1,  0,  0,  1
```

See Section 5.1.163 [uressolve], page 283.

5.1.98 mres

Syntax: `mres` (ideal_expression, int_expression)
`mres` (module_expression, int_expression)

Type: resolution

Purpose: computes a minimal free resolution of an ideal or module M with the standard basis method. More precisely, let A=`matrix`(M), then `mres` computes a free resolution of $coker(A) = F_0/M$

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0,$$

where the columns of the matrix A_1 are a minimal set of generators of M if the basering is local or if M is homogeneous. If the int expression k is not zero, then the computation stops after k steps and returns a list of modules $M_i = \texttt{module}(A_i)$, i=1...k.
`mres(M,0)` returns a resolution consisting of at most n+2 modules, where n is the number of variables of the basering. Let `list L=mres(M,0);` then `L[1]` consists of a minimal set of generators of the input, `L[2]` consists of a minimal set of generators for the first syzygy module of `L[1]`, etc., until `L[p+1]`, such that `L[i]` $\neq 0$ for $i \leq p$, but `L[p+1]`, the first syzygy module of `L[p]`, is 0 (if the basering is not a qring).

Note: Accessing single elements of a resolution may require some partial computations to be finished and may therefore take some time.

Example:

```
  ring r=31991,(t,x,y,z,w),ls;
  ideal M=t2x2+tx2y+x2yz,t2y2+ty2z+y2zw,
          t2z2+tz2w+xz2w,t2w2+txw2+xyw2;
  resolution L=mres(M,0);
  L;
↦  1      4      15      18      7      1
↦ r <--  r <--  r <--   r <--   r <--  r
↦
↦ 0      1      2       3       4      5
↦
  // projective dimension of M is 5
```

See Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 5.1.83 [lres], page 215; Section 4.14 [module], page 112; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.99 mres_map

Syntax: `mres_map (` ideal_expression`,` int_expression `)`
`mres_map (` module_expression`,` int_expression `)`

Type: resolution

Purpose: computes a minimal free resolution of an ideal or module M with the standard basis method. More precisely, let A=`matrix`(M), then `mres_map` computes a free resolution of $coker(A) = F_0/M$

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0,$$

where the columns of the matrix A_1 are a minimal set of generators of M if the basering is local or if M is homogeneous. If the int expression k is not zero, then the computation stops after k steps and returns a list of modules $M_i = \texttt{module}(A_i)$, i=1...k.
`mres_map(M,0,T)` returns a resolution consisting of at most n+2 modules, where n is

the number of variables of the basering. Let `list L=mres_map(M,0,T);` then `L[1]` consists of a minimal set of generators of the input, `L[2]` consists of a minimal set of generators for the first syzygy module of `L[1]`, etc., until `L[p+1]`, such that `L[i]` $\neq 0$ for $i \leq p$, but `L[p+1]`, the first syzygy module of `L[p]`, is 0 (if the basering is not a qring).

Note: Accessing single elements of a resolution may require some partial computations to be finished and may therefore take some time.

Example:

```
  ring r=31991,(t,x,y,z,w),ls;
  ideal M=t2x2+tx2y+x2yz,t2y2+ty2z+y2zw,
          t2z2+tz2w+xz2w,t2w2+txw2+xyw2;
  smatrix T;
  resolution L=mres_map(M,0,T);
  L;
↦   1      4      15      18      7      1
↦  r <--  r <--  r <--   r <--   r <--  r
↦
↦  0      1      2       3       4      5
↦
  // projective dimension of M is 5
  print(T);
↦  0,0,0,1,
↦  1,0,0,0,
↦  0,1,0,0,
↦  0,0,1,0
```

See Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 5.1.83 [lres], page 215; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.100 mstd

Syntax: `mstd` (ideal_expression)
`mstd` (module_expression)

Type: list

Purpose: returns a list whose first entry is a standard basis for the ideal, resp. module, whose second entry is a generating set for the ideal, resp. module. If the coefficient ring is a field and either the ideal/module is homogeneous or the ordering is local, this second entry is a minimal generating set.

Example:

```
  ring r=0,(x,y,z,t),dp;
  poly f=x3+y4+z6+xyz;
  ideal j=jacob(f),f;
  j=homog(j,t);j;
↦  j[1]=3x2+yz
↦  j[2]=4y3+xzt
↦  j[3]=6z5+xyt3
↦  j[4]=0
↦  j[5]=z6+y4t2+x3t3+xyzt3
  mstd(j);
```

```
↦ [1]:
↦    _[1]=3x2+yz
↦    _[2]=4y3+xzt
↦    _[3]=6z5+xyt3
↦    _[4]=xyzt3
↦    _[5]=y2z2t3
↦    _[6]=yz3t4
↦    _[7]=xz3t4
↦    _[8]=yz2t7
↦    _[9]=xz2t7
↦    _[10]=y2zt7
↦    _[11]=xy2t7
↦ [2]:
↦    _[1]=3x2+yz
↦    _[2]=4y3+xzt
↦    _[3]=6z5+xyt3
↦    _[4]=xyzt3
```

See Section 4.6 [ideal], page 80; Section 5.1.91 [minbase], page 221; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271.

5.1.101 mult

Syntax: `mult` (ideal_expression)
`mult` (module_expression)

Type: int

Purpose: computes the degree of the monomial ideal, resp. module, generated by the leading monomials of the input.
If the input is a standard basis of a homogeneous ideal then it returns the degree of this ideal.
If the input is a standard basis of an ideal in a (local) ring with respect to a local degree ordering then it returns the multiplicity of the ideal (in the sense of Samuel, with respect to the maximal ideal).

Example:

```
  ring r=32003,(x,y),ds;
  poly f=(x3+y5)^2+x2y7;
  ideal i=std(jacob(f));
  mult(i);
↦ 46
  mult(std(f));
↦ 6
```

See Section 5.1.20 [degree], page 171; Section 5.1.25 [dim], page 174; Section 4.6 [ideal], page 80; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

5.1.102 nameof

Syntax: `nameof` (expression)

Type: string

Purpose: returns the name of an expression as string.

Example:

```
  int i=9;
  string s=nameof(i);
  s;
↦ i
  nameof(s);
↦ s
  nameof(i+1); //returns the empty string:
↦
  nameof(basering);
↦ basering
  basering;
↦    ? `basering` is undefined
↦    ? error occurred in or before ./examples/nameof.sing line 7: `  baser
   ng;`
  ring r;
  nameof(basering);
↦ r
```

See Section 5.1.103 [names], page 229; Section 5.1.135 [reservedName], page 254; Section 5.1.161 [typeof], page 282.

5.1.103 names

Syntax: `names ( )`
`names (` ring_name `)`
`names (` package_name `)`
`names (` level `)`

Type: list of strings

Purpose: returns the names of all user-defined variables which are ring independent (this includes the names of procedures) or, in the second case, which belong to the given ring. The third case restricts the variables to the given level.

package_name can be `Current`, `Top` or any other identifier of type package.

Example:

```
  int i=9;
  ring r;
  poly f;
  package p;
  int j; exportto(p,j);
  poly g;
  setring r;
  list l=names();
  l[1..3];
↦ l p r
  names(r);
↦ [1]:
↦    g
↦ [2]:
↦    f
  names(p);
↦ [1]:
```

```
↦    j
  names(0);
↦ [1]:
↦    l
↦ [2]:
↦    p
↦ [3]:
↦    r
↦ [4]:
↦    i
↦ [5]:
↦    mathicgb_prOrder
↦ [6]:
↦    mathicgb
↦ [7]:
↦    Singmathic
↦ [8]:
↦    create_ring
↦ [9]:
↦    min
↦ [10]:
↦    max
↦ [11]:
↦    datetime
↦ [12]:
↦    weightKB
↦ [13]:
↦    fprintf
↦ [14]:
↦    printf
↦ [15]:
↦    sprintf
↦ [16]:
↦    quotient4
↦ [17]:
↦    quotient5
↦ [18]:
↦    quotient3
↦ [19]:
↦    quotient2
↦ [20]:
↦    quotient1
↦ [21]:
↦    quot
↦ [22]:
↦    res
↦ [23]:
↦    groebner
↦ [24]:
↦    qslimgb
↦ [25]:
↦    hilbRing
↦ [26]:
```

```
↦    par2varRing
↦ [27]:
↦    quotientList
↦ [28]:
↦    stdhilb
↦ [29]:
↦    stdfglm
↦ [30]:
↦    Standard
↦ [31]:
↦    Float
↦ [32]:
↦    crossprod
↦ [33]:
↦    ZZ
↦ [34]:
↦    QQ
↦ [35]:
↦    Top
```

See Section 5.1.102 [nameof], page 228; Section 5.1.135 [reservedName], page 254.

5.1.104 ncols

Syntax: `ncols` (matrix_expression)
`ncols` (smatrix_expression)
`ncols` (intmat_expression)
`ncols` (ideal_expression)

Type: int

Purpose: returns the number of columns of a matrix, an intmat, or the number of given generators of the ideal, including zeros.

Note: `size(ideal)` counts the number of generators which are different from zero. (Use `nrows` to get the number of rows of a given matrix or intmat.)

Example:

```
  ring r;
  matrix m[5][6];
  ncols(m);
↦ 6
  ideal i=x,0,y;
  ncols(i);
↦ 3
  size(i);
↦ 2
```

See Section 4.13 [matrix], page 108; Section 5.1.107 [nrows], page 232; Section 5.1.144 [size], page 264; Section 4.21 [smatrix], page 129.

5.1.105 npars

Syntax: `npars` (ring_name)

Type: int

Purpose: returns the number of parameters of a ring.

Example:

```
  ring r=(23,t,v),(x,a(1..7)),lp;
  // the parameters are t,v
  npars(r);
↦ 2
```

See Section 5.1.114 [par], page 239; Section 5.1.116 [parstr], page 240; Section 4.20 [ring], page 127.

5.1.106 nres

Syntax: `nres` (ideal_expression, int_expression)
`nres` (module_expression, int_expression)

Type: resolution

Purpose: computes a free resolution of an ideal or module M which is minimized from the second module on (by the standard basis method).

More precisely, let A_1=matrix(M), then `nres` computes a free resolution of $coker(A_1) = F_0/M$

$$\ldots \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0,$$

where the columns of the matrix A_1 are the given set of generators of M. If the int expression k is not zero then the computation stops after k steps and returns a list of modules $M_i = \texttt{module}(A_i)$, $i = 1, \ldots, k$.

`nres(M,0)` returns a list of n modules where n is the number of variables of the basering. Let `list L=nres(M,0);` then `L[1]=M` is identical to the input, `L[2]` is a minimal set of generators for the first syzygy module of `L[1]`, etc. (L[i] $= M_i$ in the notations from above).

Example:

```
  ring r=31991,(t,x,y,z,w),ls;
  ideal M=t2x2+tx2y+x2yz,t2y2+ty2z+y2zw,
          t2z2+tz2w+xz2w,t2w2+txw2+xyw2;
  resolution L=nres(M,0);
  L;
↦  1      4      15      18      7      1
↦ r <--  r <--  r <--   r <--   r <--  r
↦
↦ 0      1      2       3       4      5
↦ resolution not minimized yet
↦
```

See Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 5.1.83 [lres], page 215; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.134 [res], page 253; Section 4.19 [resolution], page 125; Section 5.1.149 [sres], page 269.

5.1.107 nrows

Syntax: `nrows` (matrix_expression)
`nrows` (smatrix_expression)
`nrows` (intmat_expression)

`nrows` (intvec_expression)
`nrows` (module_expression)
`nrows` (vector_expression)

Type: int

Purpose: returns the number of rows of a matrix, an intmat or an intvec, resp. the minimal rank of a free module in which the given module or vector lives (the index of the last non-zero component).

Note: Use `ncols` to get the number of columns of a given matrix or intmat.

Example:

```
  ring R;
  matrix M[2][3];
  nrows(M);
↦ 2
  nrows(freemodule(4));
↦ 4
  module m=[0,0,1];
  nrows(m);
↦ 3
  nrows([0,x,0]);
↦ 2
```

See Section 5.1.51 [gen], page 191; Section 4.13 [matrix], page 108; Section 4.14 [module], page 112; Section 5.1.104 [ncols], page 231; Section 4.21 [smatrix], page 129; Section 4.23 [vector], page 134.

5.1.108 numerator

Syntax: `numerator` (number_expression)

Type: number

Purpose: returns the numerator of a number.

Example:

```
  ring r = 0, x, dp;
  number n = 3/2;
  numerator(n);
↦ 3
```

See Section 5.1.9 [cleardenom], page 163; Section D.2.8.14 [content], page 894; Section 5.1.22 [denominator], page 172.

5.1.109 nvars

Syntax: `nvars` (ring_name)

Type: int

Purpose: returns the number of variables of a ring.

Example:

```
  ring r=(23,t,v),(x,a(1..7)),ls;
  // the variables are x,a(1),...,a(7)
  nvars(r);
↦ 8
```

See Section 5.1.105 [npars], page 231; Section 4.20 [ring], page 127; Section 5.1.165 [var], page 285; Section 5.1.167 [varstr], page 285.

5.1.110 open

Syntax: `open` (link_expression)

Type: none

Purpose: opens a link.

Example:

```
link l="ssi:tcp localhost:"+system("Singular");
open(l); // start SINGULAR "server" on localhost in batchmode
close(l); // shut down SINGULAR server
```

See Section 5.1.10 [close], page 163; Section 4.10 [link], page 96.

5.1.111 option

Syntax: `option ()`

Type: string

Purpose: lists all defined options.

Syntax: `option` (option_name)

Type: none

Purpose: sets an option.

Note: To disable an option, use the prefix `no`.

Syntax: `option ( get )`

Type: intvec

Purpose: dumps the state of all options to an intvec.

Syntax: `option ( set,` intvec_expression)

Type: none

Purpose: restores the state of all options from an intvec (produced by `option(get)`).

Values: The following options are used to manipulate the behavior of computations and act like boolean switches. Use the prefix `no` to disable an option. Notice that some options are ring dependent and reset to their default values on a change of the current basering.

`none` turns off all options (including the `prompt` option).

`warn` be aware of pitfalls. See Section 3.9.7 [option(warn)], page 71.

`returnSB` the functions `syz`, `intersect` (2 arguments), `quotient` return a standard base instead of a generating set if `returnSB` is set. This option should not be used for `lift`.

`fastHC` tries to find the highest corner of the staircase (HC) as fast as possible during a standard basis computation (only used for local orderings).

`infRedTail`
local normal form computations will not use the ecart to avoid possibly infinite tail reductions: should only be used with extreme care.
By default, it is only set in the case of a zero-dimensional ideal.

`intStrategy`
avoids division of coefficients during standard basis computations. This option is ring dependent. By default, it is set for rings with characteristic 0 and not set for all other rings.

`lazy` uses a more lazy approach in std computations, which was used in Singular version before 2-0 (and which may lead to faster or slower computations, depending on the example)

`length` select shorter reducers in std computations,

`notRegularity`
disables the regularity bound for `res` and `mres` (see Section 5.1.132 [regularity], page 252).

`notSugar` turns off sugar strategy during standard basis computation and reduction.

`notBuckets`
disables the bucket representation of polynomials during standard basis computations. This option usually decreases the memory consumption but increases the computation time. It should only be set for memory-critical standard basis computations.

`prot` shows protocol information indicating the progress during the following computations: `facstd`, `fglm`, `groebner`, `intersect`, `lres`, `mres`, `minres`, `mstd`, `res`, `slimgb`, `sres`, `std`, `stdfglm`, `stdhilb`, `syz`. See below for more details.

`qringNF` simplifies modulo the current `qring` in all assignments.

`redSB` computes a reduced standard basis in any standard basis computation (in rings with global ior local orderings, See Section 5.1.64 [interred], page 201 for the discussion of reduced for local orderings))

`redTail` reduction of the tails of polynomials during standard basis computations. This option is ring dependent. By default, it is set for rings with global degree orderings and not set for all other rings. This option changes the reduction strategy and may decrease/increase time and memory consumption - it does not ensure tail reduction on the result - use redSB for that.

`redThrough`
for inhomogeneous input, polynomial reductions during standard basis computations are never postponed, but always finished through. This option is ring dependent. By default, it is set for rings with global degree orderings and not set for all other rings. This option changes the reduction strategy and may decrease/increase time and memory consumption.

`sugarCrit`
: uses criteria similar to the homogeneous case to keep more pairs which would be excluded by other criteria but which may be useful for downstream computations. This option changes the strategy for criteria and selection and may decrease/increase time and memory consumption.

`weightM` automatically computes suitable weights for the weighted ecart and the weighted sugar method.

`cancelunit`
: avoids to divide polynomials by non-constant units in `std` in the local case. Should usually not be used.

`contentSB`
: avoids to divide by the content of a polynomial in `std` and related algorithms. Should usually not be used.

`intersectElim`
: prefers elimination to compute intersections (experimental, will be removed in the next release). Should usually not be used.

`intersectSyz`
: prefers syzygy methods to compute intersections (experimental, will be removed in the next release). Should usually not be used.

The following options, which also control computations, are special, since they are not manipulated by the `option` command but by a direct assignment of a value. Reset the option by assigning the value 0; the command `option(none)` will not reset them! If there is a non-zero value assigned, the command `option()` prints the option.

`multBound`
: a multiplicity bound is set (see Section 5.3.4 [multBound], page 304).

`degBound` a degree bound is set (see Section 5.3.1 [degBound], page 302).

The last set of options controls the output of Singular:

`Imap` shows the mapping of variables with the fetch commands.

`debugLib` warns about syntax errors when loading a library.

`defRes` shows the names of the syzygy modules while converting `resolution` to `list`

`loadLib` shows loading of libraries (set by default).

`loadProc` shows loading of procedures from libraries.

`mem` shows memory usage in square brackets (see Section 5.1.89 [memory], page 219).

`notWarnSB`
: do not warn about using a generating set instead of a standard basis.

`prompt` shows prompt (`>`, resp. `.`) if ready for input (default).

`reading` shows the number of characters read from a file.

`redefine` warns about variable redefinitions (set by default).

`usage`   shows correct usage in error messages (set by default).

Example:

```
  option(prot);
  option();
↦ //options: prot redefine usage prompt
  option(notSugar);
  option();
↦ //options: prot notSugar redefine usage prompt
  option(noprot);
  option();
↦ //options: notSugar redefine usage prompt
  option(none);
  option();
↦ //options: none
  ring r=0,x,dp;
  degBound=22;
  option();
↦ //options: degBound redTail redThrough intStrategy
  intvec i=option(get);
  option(none);
  option(set,i);
  option();
↦ //options: degBound redTail redThrough intStrategy
```

The output reported on `option(prot)` has the following meaning:

(command)	(character)	(meaning)
`facstd`	`F`	found a new factor
		all other characters: like the output of `std` and `reduce`
`fglm`	`.`	basis monomial found
	`+`	edge monomial found
	`-`	border monomial found
`groebner`		all characters: like the output of `std/slimgb`
`lres`	`.`	minimal syzygy found
	n	slanted degree, i.e., row of Betti matrix
	`(mn)`	calculate in module n
	`g`	pair found giving reductum and syzygy
`mres`	`[d]`	computations of the d-th syzygy module
		all other characters: like the output of `std`
`minres`	`[d]`	minimizing of the d-th syzygy module
`mstd`		all characters: like the output of `std`
`reduce`	`r`	reduced a leading term
	`t`	reduced a non-leading term

`res`	`[d]`	computations of the d-th syzygy module all other characters: like the output of `std`
`slimgb`	`M[n,m]`	parallel reduction of n elements with m non-zero output elements
	`v`	candidate for postponing, need to canonicalize
	`.`	postponed a reduction of a pair/S-polynomial
	`b`	exchange of a reductor by a 'better' one
	`e`	a new reductor with non-minimal leading term
	`r`	redTail reduction
	`n`	no redTail reduction
	`B`	resort pairs
	`C`	slimgb_alg::cleanDegs
	`(n)`	n critical pairs are still to be reduced
	d	the maximal degree of the leading terms is currently d
`sres`	`.`	syzygy found
	`(n)`	n elements remaining
	`[n]`	finished module n
`std`	`[m:n]`	internal ring change to polynomial representation with exponent bound m and n words in exponent vector
	`s`	found a new element of the standard basis
	`-`	reduced a pair/S-polynomial to 0
	`.`	postponed a reduction of a pair/S-polynomial
	`h`	used Hilbert series criterion
	`H(d)`	found a 'highest corner' of degree d, no need to consider higher degrees
	`(n)`	n critical pairs are still to be reduced
	`(S:n)`	doing complete reduction of n elements
	d	the degree of the leading terms is currently d
`stdfglm`		all characters in first part: like the output of `std` all characters in second part: like the output of `fglm`
`stdhilb`		all characters: like the output of `std`
`syz`		all characters: like the output of `std`

See Section 5.3.1 [degBound], page 302; Section 5.3.4 [multBound], page 304; Section 5.1.151 [std], page 271.

5.1.112 ord

Syntax: `ord (` poly_expression `)`
`ord (` vector_expression `)`

Type: int

Purpose: returns the (weighted) degree of the initial term of a polynomial or a vector; the weights are the weights used for the first block of the ring ordering.

Note: `ord(0)` is `-1`.
In a global degree ordering `ord` is the same as `deg`.

Example:

```
  ring r=7,(x,y),wp(2,3);
  ord(0);
↦ -1
  poly f=x2+y3;  // weight on y is 3
  ord(f),deg(f);
↦ 9 9
  ring R=7,(x,y),ws(2,3);
  poly f=x2+y3;
  ord(f),deg(f);
↦ 4 9
  vector v=[x2,y];
  ord(v),deg(v);
↦ 3 4
```

See Section 5.1.19 [deg], page 170; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.113 ordstr

Syntax: `ordstr` (ring_name)

Type: string

Purpose: returns the description of the monomial ordering of the ring.

Example:

```
  ring r=7,(x,y),wp(2,3);
  ordstr(r);
↦ wp(2,3),C
```

See Section 5.1.7 [charstr], page 162; Section 5.1.116 [parstr], page 240; Section 4.20 [ring], page 127; Section 5.1.167 [varstr], page 285.

5.1.114 par

Syntax: `par` (int_expression)

Type: number

Purpose: `par(n);` returns the n-th parameter of the basering.

Example:

```
  ring r=(0,a,b,c),(x,y,z),dp;
  char(r);  // char to get the characteristic
↦ 0
  par(2);   // par to get the n-th parameter
↦ (b)
```

See Section 5.1.5 [char], page 161; Section 5.1.105 [npars], page 231; Section 5.1.116 [parstr], page 240; Section 4.20 [ring], page 127; Section 5.1.165 [var], page 285.

5.1.115 pardeg

Syntax: `pardeg` (number_expression)

Type: int

Purpose: returns the degree of a number considered as a polynomial in the ring parameters.

Example:

```
  ring r=(0,a,b,c),(x,y,z),dp;
  pardeg(a^2*b);
↦ 3
```

See Section 5.1.19 [deg], page 170; Section 4.15 [number], page 115; Section 4.20 [ring], page 127; Section 5.1.165 [var], page 285.

5.1.116 parstr

Syntax: `parstr` (ring_name)
`parstr` (int_expression)
`parstr` (ring_name, int_expression)

Type: string

Purpose: returns the list of parameters of the ring as a string or the name of the n-th parameter where n is given by the int_expression.
If the ring_name is omitted, the basering is used, thus `parstr(n)` is equivalent to `parstr(basering,n)`.

Example:

```
  ring r=(7,a,b,c),(x,y),wp(2,3);
  parstr(r);
↦ a,b,c
  parstr(2);
↦ b
  parstr(r,3);
↦ c
```

See Section 5.1.7 [charstr], page 162; Section 5.1.105 [npars], page 231; Section 5.1.113 [ordstr], page 239; Section 4.20 [ring], page 127; Section 5.1.167 [varstr], page 285.

5.1.117 preimage

Syntax: `preimage` (map)
`preimage` (ring_name, map_name, ideal_name)
`preimage` (ring_name, ideal_expression, ideal_name)

Type: ring
ideal

Purpose: returns the source ring of a map (in the first case) or returns the preimage of an ideal under a given map.
The second argument has to be a map from the basering to the given ring (or an ideal defining such a map), and the ideal has to be an ideal in the given ring.

Note: As `preimage` is handling ideals (not polynomials), the result of a preimage calculation of a principal ideal is (the closure of) the preimage of the ideal, not that of the polynomial.

Example:

```
  ring r1=32003,(x,y,z,w),lp;
  ring r=32003,(x,y,z),dp;
  ideal i=x,y,z;
  ideal i1=x,y;
  ideal i0=0;
  map f=r1,i;
  nameof (preimage (f));
↦ r1
  setring r1;
  ideal i1=preimage(r,f,i1);
  i1;
↦ i1[1]=w
↦ i1[2]=y
↦ i1[3]=x
  // the kernel of f
  preimage(r,f,i0);
↦ _[1]=w
  // or, use:
  kernel(r,f);
↦ _[1]=w
```

See Section 4.6 [ideal], page 80; Section 5.1.70 [kernel], page 206; Section 4.12 [map], page 106; Section 4.20 [ring], page 127.

5.1.118 prime

Syntax: `prime` (int_expression)

Type: int

Purpose: returns the largest prime less than or equal to the argument; returns 2 for all arguments smaller than 3.

Example:

```
  prime(320000);
↦ 319993
  prime(32004);
↦ 32003
  prime(0);
↦ 2
  prime(-1);
↦ 2
```

See Section D.2.3 [general_lib], page 805; Section 4.7 [int], page 85.

5.1.119 primefactors

Syntax: `primefactors` (int/bigint/number_expression)
`primefactors` (int/bigint/number_expression , int_expression)

Type: list

Purpose: returns the prime factorisation up to an optionally given bound, b, on the prime factors
When called with int(s)/bigint(s), no ring needs to be active.
When called with numbers these are assumed to be integers in a polynomial ring over Q.
The method finds all prime factors of an integer n. n' will contain the sign, be zero, or the rest (when a bound is given) respectively. The returned list contains the following

information: The returned list contains the following information:
L[1][i] = i-th prime factor (in ascending order),
L[2][i] = multiplicity of L[1][i],
L[3] = n'

Example:

```
  bigint n = bigint(7)^12 * bigint(37)^6 * 121;
  primefactors(n);
↦ [1]:
↦    [1]:
↦       7
↦    [2]:
↦       11
↦    [3]:
↦       37
↦ [2]:
↦    [1]:
↦       12
↦    [2]:
↦       2
↦    [3]:
↦       6
↦ [3]:
↦    1
  primefactors(n,25);
↦ [1]:
↦    [1]:
↦       7
↦    [2]:
↦       11
↦ [2]:
↦    [1]:
↦       12
↦    [2]:
↦       2
↦ [3]:
↦    2565726409
```

See Section 5.1.118 [prime], page 241.

5.1.120 print

Syntax: `print` (expression)
`print` (expression`,` `"betti"`)
`print` (expression`,` format_string)

Type: string

Purpose: The first form prints the expression.
The second form prints the graded Betti numbers from a matrix. The Betti numbers are printed in a matrix-like format where the entry d in row i and column j is the minimal number of generators in degree $i+j$ of the j-th syzygy module of R^n/M (the 0th and 1st syzygy module of R^n/M is R^n and M, resp.).
The last form returns the printed output as a string depending on the format string iwhich determines the format to use to generate the string.

The following format strings are supported:

`"%s"` returns `string(` expression `)`,

`"%2s"` similar to `"%s"`, except that newlines are inserted after every comma and at the end,

`"%l"` similar to `"%s"`, except that each object is embraced by its type such that it can be directly used for "cutting and pasting",

`"%2l"` similar to `"%l"`, except that newlines are inserted after every comma and at the end,

`"%;"` returns the string equivalent to typing `expression;`

`"%t"` returns the string equivalent to typing `type expression;`

`"%p"` returns the string equivalent to typing `print(expression);`

`"%b"` returns the string equivalent to typing `print(expression, "betti");`

`"betti"` is not a format string.

Example:

```
  ring r=0,(x,y,z),dp;
  module m=[1,y],[0,x+z];
  m;
↦ m[1]=y*gen(2)+gen(1)
↦ m[2]=x*gen(2)+z*gen(2)
  print(m);  // the columns generate m
↦ 1,0,
↦ y,x+z
  string s=print(m,"%s"); s;
↦ y*gen(2)+gen(1),x*gen(2)+z*gen(2)
  s=print(m,"%2s"); s;
↦ y*gen(2)+gen(1),
↦ x*gen(2)+z*gen(2)
↦ 
  s=print(m,"%l"); s;
↦ module(y*gen(2)+gen(1),x*gen(2)+z*gen(2))
  s=print(m,"%;"); s;
↦ m[1]=y*gen(2)+gen(1)
↦ m[2]=x*gen(2)+z*gen(2)
↦ 
  s=print(m,"%t"); s;
↦ // m module , rk 2
↦ m[1]=y*gen(2)+gen(1)
↦ m[2]=x*gen(2)+z*gen(2)
  s=print(m,"%p"); s;
↦ 1,0,
↦ y,x+z
  intmat M=betti(mres(m,0));
  print(M,"betti");
↦            0     1
↦ ------------------
↦     0:     1     1
↦ ------------------
↦ total:     1     1
```

```
↦
  list l=r,M;
  s=print(l,"%s"); s;
↦ (QQ),(x,y,z),(dp(3),C),1,1
  s=print(l,"%2s"); s;
↦ (QQ),(x,y,z),(dp(3),C),
↦ 1,1
↦
  s=print(l,"%l"); s;
↦ list("(QQ),(x,y,z),(dp(3),C)",intmat(intvec(1,1 ),1,2))
```

See Section 3.5.5 [Type conversion and casting], page 46; Section 5.1.4 [betti], page 159; Section 5.1.17 [dbprint], page 169; Section 5.1.46 [fprintf], page 186; Section 5.1.121 [printf], page 244; Section 5.3.7 [short], page 305; Section 5.1.148 [sprintf], page 267; Section 4.22.3 [string type cast], page 131; Section 5.1.160 [type], page 282.

5.1.121 printf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `printf (` string_expression `[,` any_expressions`] )`

Return: none

Purpose: `printf(fmt,...);` performs output formatting. The first argument is a format control string. Additional arguments may be required, depending on the content of the control string. A series of output characters is generated as directed by the control string; these characters are displayed (i.e., printed to standard out).
The control string `fmt` is simply text to be copied, except that the string may contain conversion specifications.
Type `help print;` for a listing of valid conversion specifications. As an addition to the conversions of `print`, the `%n` and `%2` conversion specification does not consume an additional argument, but simply generates a newline character.

Note: If one of the additional arguments is a list, then it should be enclosed once more into a `list()` command, since passing a list as an argument flattens the list by one level.

Example:

```
  ring r=0,(x,y,z),dp;
module m=[1,y],[0,x+z];
intmat M=betti(mres(m,0));
list l=r,m,matrix(M);
printf("s:%s,l:%l",1,2);
↦ s:1,l:int(2)
printf("s:%s",l);
↦ s:(QQ),(x,y,z),(dp(3),C)
printf("s:%s",list(l));
↦ s:(QQ),(x,y,z),(dp(3),C),y*gen(2)+gen(1),x*gen(2)+z*gen(2),1,1
printf("2l:%2l",list(l));
↦ 2l:list("(QQ),(x,y,z),(dp(3),C)",
↦ module(y*gen(2)+gen(1),
↦ x*gen(2)+z*gen(2)),
↦ matrix(ideal(1,
↦ 1),1,2))
↦
```

```
printf("%p",matrix(M));
↦ 1,1
printf("%;",matrix(M));
↦ _[1,1]=1
↦ _[1,2]=1
↦
printf("%b",M);
↦            0     1
↦ ------------------
↦     0:     1     1
↦ ------------------
↦ total:     1     1
↦
```

See also: Section 5.1.46 [fprintf], page 186; Section 5.1.120 [print], page 242; Section 5.1.148 [sprintf], page 267; Section 4.22 [string], page 130.

5.1.122 prune

Syntax: `prune` (module_expression)

Type: module

Purpose: returns the module minimally embedded in a free module such that the corresponding factor modules are isomorphic.

Note: If for the input module the attribute `"isHomog"` is set, `prune` also sets the attribute `"isHomog"`.
For non-global orderings, only reduction steps with constant units are performed. Hence, the returned module does not need to be minimal.

Note: `The coefficiensts must be a field.`

Example:

```
  ring r=0,(x,y,z),dp;
  module m=gen(1),gen(3),[x,y,0,z],[x+y,0,0,0,1];
  print(m);
↦ 1,0,x,x+y,
↦ 0,0,y,0,
↦ 0,1,0,0,
↦ 0,0,z,0,
↦ 0,0,0,1
  print(prune(m));
↦ y,
↦ z
```

See Section 4.14 [module], page 112.

5.1.123 prune_map

Syntax: `prune_map(` module_expression , smatrix_name)

Type: module

Purpose: returns the module minimally embedded in a free module such that the corresponding factor modules are isomorphic together with the map (as matrix)

Note: If for the input module the attribute `"isHomog"` is set, `prune` also sets the attribute `"isHomog"`.
For non-global orderings, only reduction steps with constant units are performed. Hence, the returned module does not need to be minimal.

Example:

```
  ring r=0,(x,y,z),dp;
  module m=gen(1),gen(3),[x,y,0,z],[x+y,0,0,0,1];
  smatrix M;
  print(m);
↦ 1,0,x,x+y,
↦ 0,0,y,0,
↦ 0,1,0,0,
↦ 0,0,z,0,
↦ 0,0,0,1
  print(prune_map(m,M)); // pruned module
↦ 0,
↦ y,
↦ 0,
↦ z,
↦ 0
  print(m*M); // apply map
↦ 0,
↦ y,
↦ 0,
↦ z,
↦ 0
  print(M); // map
↦ -x,
↦ 0,
↦ 1,
↦ 0
```

See Section 4.14 [module], page 112.

5.1.124 qhweight

Syntax: `qhweight` (ideal_expression)

Type: intvec

Purpose: computes the weight vector of the variables for a quasihomogeneous ideal. If the input is not weighted homogeneous, an intvec of zeros is returned.

Example:

```
  ring h1=32003,(t,x,y,z),dp;
  ideal i=x4+y3+z2;
  qhweight(i);
↦ 0,3,4,6
```

See Section 4.6 [ideal], page 80; Section 4.9 [intvec], page 93; Section 5.1.172 [weight], page 288.

5.1.125 qrds

Syntax: `qrds (` matrix_expression`,` number_expression`,` number_expression`,` number_expression`)`

Type: list

Purpose: computes all eigenvalues with multiplicities of the given matrix by performing the numeric QR double shift algorithm involving Hessenberg form and householder transformations.
This method expects the ground field to be the complex numbers, and all matrix entries to be real numbers, i.e., elements of this ground field with the imaginary part equal to zero.
If the algorithm works, then it returns a list with two entries which are again lists of the same size:
_[1][i] is the i-th mutually distinct eigenvalue that was found,
_[2][i] is the (int) multiplicity of _[1][i].
If the algorithm does not work (due to an ill-posed matrix), a list with the single entry (int)0 is returned.
The first number argument is used for detection of deflation in the actual QR double shift algorithm. The second number argument is used for ending Heron's iteration whenever square roots are being computed. And the third number argument is used to distinguish between distinct eigenvalues: When the Euclidean distance between two computed eigenvalues is less then this number, then they will be regarded equal, resulting in a higher multiplicity of the corresponding eigenvalue. (A good choice for all three number arguments is a small value like e.g. 10^(-100).)

Example:

```
  ring r=(complex,50),(dummy),dp;
  matrix A[3][3]=-10,37,-5,-14,51,-10,-29,99,-18;
  bigint b = bigint(10)^100; number t = 1/b;
  list L=qrds(A,t,t,t); L;
↦ [1]:
↦    [1]:
↦ (3+i*2)
↦    [2]:
↦ (3-i*2)
↦    [3]:
↦ 17
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
```

5.1.126 quote

Syntax: `quote (` expression `)`

Type: none

Purpose: prevents expressions from evaluation. Used only in connections with write to ssi links, prevents evaluation of an expression before sending it to an other SINGULAR process. Within a quoted expression, the quote can be "undone" by an `eval` (i.e., each eval "undoes" the effect of exactly one quote).

Example:

```
  link l="ssi:w example.ssi";
  ring r=0,(x,y,z),ds;
  ideal i=maxideal(3);
  ideal j=x7,x2,z;
  option(prot);
  // compute i+j before writing, but not std
  write (l, quote(std(eval(i+j))));
  close(l);
  // now read it in again and evaluate:
  read(l);
↦ [1048575:1]1(12)s2(11)s3(10)--s(7)s(6)-----7-
↦ product criterion:4 chain criterion:0
↦ _[1]=z
↦ _[2]=x2
↦ _[3]=xy2
↦ _[4]=y3
  close(l);
```

See Section 4.10.5 [Ssi links], page 99; Section 5.1.29 [eval], page 177; Section 5.1.174 [write], page 289.

5.1.127 quotient

Syntax: `quotient (` ideal_expression`,` ideal_expression `)`
`quotient (` module_expression`,` module_expression `)`

Type: ideal

Syntax: `quotient (` module_expression`,` ideal_expression `)`

Type: module

Purpose: computes the ideal quotient, resp. module quotient. Let `R` be the basering, `I`,`J` ideals and `M` a module in R^n. Then

`quotient(I,J)`$= \{a \in R \mid aJ \subset I\}$,

`quotient(M,J)`$= \{b \in R^n \mid bJ \subset M\}$.

Example:

```
ring r=181,(x,y,z),(c,ls);
ideal id1=maxideal(3);
ideal id2=x2+xyz,y2-z3y,z3+y5xz;
ideal id6=quotient(id1,id2);
id6;
↦ id6[1]=z
↦ id6[2]=y
↦ id6[3]=x
quotient(id2,id1);
↦ _[1]=z2
↦ _[2]=yz
↦ _[3]=y2
↦ _[4]=xz
↦ _[5]=xy
↦ _[6]=x2
module m=x*freemodule(3),y*freemodule(2);
```

```
ideal id3=x,y;
quotient(m,id3);
↦ _[1]=[1]
↦ _[2]=[0,1]
↦ _[3]=[0,0,x]
```

See Section 5.1.40 [fglmquot], page 184; Section 4.6 [ideal], page 80; Section D.4.14.1 [modQuotient], page 1120; Section 4.14 [module], page 112.

5.1.128 random

Syntax: `random` (int_expression, int_expression)

Type: int

Purpose: returns a random integer between the integer given by the first int_expression and the one given by the second int_expression.

Syntax: `random` (int_expression, int_expression, int_expression)

Type: intmat

Purpose: returns a random intmat where the size is given by the second (number of rows) and third argument (number of columns). The absolute value of the entries of the matrix is smaller than or equal to the integer given as the first argument.

Note: The random generator can be set to a startvalue with the function `system`, resp. by a command line option. The current value of the random generator is `system("random")`.
Internally a random generator with values in 1 to 2^31 - 2 and a full period is used, max-min may not be larger than 2^31-2.

Example:

```
  random(1,1000);
↦ 35
  random(1,2,3);
↦ 0,0,0,
↦ 1,1,-1
  system("random",210);  // start random generator with 210
  random(-1000,1000);
↦ 707
  random(-1000,1000);
↦ 284
  system("random",210);
  random(-1000,1000);    // the same random values again
↦ 707
```

See Section 3.1.6 [Command line options], page 19; Section 4.7 [int], page 85; Section 4.8 [intmat], page 90; Section 5.1.155 [system], page 275.

5.1.129 rank

Syntax: `rank` (matrix_expression) #*`rank` (matrix_expression ,1)

Type: int

Purpose: returns the rank of a given matrix which is filled with elements of the ground field. The first variant uses a LU-decomposition, the second a row-echelon form.

Note: The function works by computing the row echelon form of the matrix using the same algorithm as for `ludecomp`.

Example:

```
ring s = 0, x, dp;
matrix A[100][100];
int i; int j; int r;
for (i = 1; i <= 100; i++)
{
  for (j = 1; j <= 100; j++)
  {
    A[i, j] = random(-10, 10);
  }
}
r = rank(A); r;
↦ 100
```

See Section 5.1.84 [ludecomp], page 216.

5.1.130 read

Syntax: `read` (link_expression)
for DBM links:
`read` (link_expression)
`read` (link_expression, string_expression)

Type: any

Purpose: reads data from a link.
For ASCII links, the content of the entire file is returned as a string. If the ASCII link is the empty string, `read` reads from standard input.
For ssi links, one expression is read from the link and returned after evaluation. See Section 4.10.5 [Ssi links], page 99.
For ssi links the `read` command blocks as long as there is no data to be read from the link. The `status` command can be used to check whether or not there is data to be read.
For DBM links, a `read` with one argument returns the value of the next entry in the data base, and a `read` with two arguments returns the value to the key given as the second argument from the data base. See Section 4.10.7 [DBM links], page 102.

Example:

```
ring r=32003,(x,y,z),dp;
ideal i=x+y,z3+22y;
// write the ideal i to the file save_i
write(":w save_i",i);
ring r0=0,(x,y,z),Dp;
// create an ideal k equal to the content
// of the file save_i
string s="ideal k="+read("save_i")+";";
execute(s);
k;
↦ k[1]=x+y
```

```
         ↦ k[2]=z3+22y
```

See Section 5.1.32 [execute], page 178; Section 5.1.52 [getdump], page 191; Section 4.10 [link], page 96; Section 5.1.150 [status], page 270; Section 5.1.174 [write], page 289.

5.1.131 reduce

Syntax: `reduce` (poly_expression, ideal_expression)
`reduce` (poly_expression, ideal_expression, int_expression)
`reduce` (poly_expression, poly_expression, ideal_expression)
`reduce` (vector_expression, ideal_expression)
`reduce` (vector_expression, ideal_expression, int_expression)
`reduce` (vector_expression, module_expression)
`reduce` (vector_expression, module_expression, int_expression)
`reduce` (vector_expression, poly_expression, module_expression)
`reduce` (ideal_expression, ideal_expression)
`reduce` (ideal_expression, ideal_expression, int_expression)
`reduce` (ideal_expression, matrix_expression, ideal_expression)
`reduce` (module_expression, ideal_expression)
`reduce` (module_expression, ideal_expression, int_expression)
`reduce` (module_expression, module_expression)
`reduce` (module_expression, module_expression, int_expression)
`reduce` (module_expression, matrix_expression, module_expression)
`reduce` (poly/vector/ideal/module, ideal/module, int, intvec)
`reduce` (ideal, matrix, ideal, int)
`reduce` (poly, poly, ideal, int)
`reduce` (poly, poly, ideal, int, intvec)

Type: the type of the first argument

Purpose: reduces a polynomial, vector, ideal or module to its normal form with respect to an ideal or module represented by a standard basis. Returns 0 if and only if the polynomial (resp. vector, ideal, module) is an element (resp. subideal, submodule) of the ideal (resp. module). The result may have no meaning if the second argument is not a standard basis.
The third (optional) argument of type int modifies the behavior:

0 default

1 consider only the leading term and do no tail reduction.

2 tail reduction:n the local/mixed ordering case: reduce also with bad ecart

4 reduce without division, return possibly a non-zero constant multiple of the remainder

If a second argument `u` of type poly or matrix is given, the first argument `p` is replaced by `p/u`. This works only for zero dimensional ideals (resp. modules) in the third argument and gives, even in a local ring, a reduced normal form which is the projection to the quotient by the ideal (resp. module). One may give a degree bound in the fourth argument with respect to a weight vector in the fifth argument in order have a finite computation. If some of the weights are zero, the procedure may not terminate!

Note: The commands `reduce` and `NF` are synonymous.

Example:

```
  ring r1 = 0,(z,y,x),ds;
  poly s1=2x5y+7x2y4+3x2yz3;
  poly s2=1x2y2z2+3z8;
  poly s3=4xy5+2x2y2z3+11x10;
  ideal i=s1,s2,s3;
  ideal j=std(i);
  reduce(3z3yx2+7y4x2+yx5+z12y2x2,j);
↦ -yx5+2401/81y14x2+2744/81y11x5+392/27y8x8+224/81y5x11+16/81y2x14
  reduce(3z3yx2+7y4x2+yx5+z12y2x2,j,1);
↦ -yx5+z12y2x2
  // 4 arguments:
  ring rs=0,x,ds;
  // normalform of 1/(1+x) w.r.t. (x3) up to degree 5
  reduce(poly(1),1+x,ideal(x3),5);
↦ // ** _ is no standard basis
↦ 1-x+x2
```

See Section 5.1.26 [division], page 174; Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 4.17.3 [poly operations], page 121; Section 5.1.151 [std], page 271; Section 4.23 [vector], page 134.

5.1.132 regularity

Syntax: `regularity` (list_expression)
`regularity` (resolution_expression)

Type: int

Purpose: computes the regularity of a homogeneous ideal, resp. module, from a minimal resolution given by the argument.
Let $0 \to \bigoplus_a K[x]e_{a,n} \to \dots \to \bigoplus_a K[x]e_{a,0} \to I \to 0$ be a minimal resolution of I considered with homogeneous maps of degree 0. The regularity is the smallest number s with the property $\deg(e_{a,i}) \leq s+i$ for all i.

Note: If applied to a non minimal resolution only an upper bound is returned.
If the input to the commands `res` and `mres` is homogeneous the regularity is computed and used as a degree bound during the computation unless `option(notRegularity);` is given.

Example:

```
  ring rh3=32003,(w,x,y,z),(dp,C);
  poly f=x11+y10+z9+x5y2+x2y2z3+xy3*(y2+x)^2;
  ideal j=homog(jacob(f),w);
  def jr=res(j,0);
  regularity(jr);
↦ 25
  // example for upper bound behaviour:
  list jj=jr;
  regularity(jj);
↦ 25
  jj=nres(j,0);
  regularity(jj);
↦ 27
  jj=minres(jj);
  regularity(jj);
```

```
↦ 25
```

See Section 5.1.48 [fres], page 188; Section 4.11 [list], page 103; Section 5.1.93 [minres], page 223; Section 5.1.98 [mres], page 225; Section 5.1.111 [option], page 234; Section 5.1.134 [res], page 253; Section 4.19 [resolution], page 125; Section 5.1.149 [sres], page 269.

5.1.133 repart

Syntax: `repart` (number_expression)

Type: number

Purpose: returns the real part of a number from a complex ground field,
returns its argument otherwise.

Example:

```
  ring r=(complex,i),x,dp;
  repart(1+2*i);
↦ 1
```

See Section 5.1.60 [impart], page 199.

5.1.134 res

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `res` (ideal_expression`,` int_expression `[,` any_expression `])`
`res` (module_expression`,` int_expression `[,` any_expression `])`

Type: resolution

Purpose: computes a (possibly minimal) free resolution of an ideal or module using a heuristically chosen method.
The second (int) argument (say `k`) specifies the length of the resolution. If it is not positive then `k` is assumed to be the number of variables of the basering.
If a third argument is given, the returned resolution is minimized.

Depending on the input, the returned resolution is computed using the following methods:

quotient rings:
`nres` (classical method using syzygies) , see Section 5.1.106 [nres], page 232.

homogeneous ideals and k=0:
`lres` (La'Scala's method), see Section 5.1.83 [lres], page 215.

not minimized resolution and (homogeneous input with k not 0, or local rings):
`sres` (Schreyer's method), see Section 5.1.149 [sres], page 269.

all other inputs:
`mres` (classical method), see Section 5.1.98 [mres], page 225.

Note: Accessing single elements of a resolution may require some partial computations to be finished and may therefore take some time.

See also Section 5.1.4 [betti], page 159; Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 5.1.83 [lres], page 215; Section 5.1.93 [minres], page 223; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.106 [nres], page 232; Section 4.19 [resolution], page 125; Section 5.1.149 [sres], page 269.

Example:

```
  ring r=0,(x,y,z),dp;
ideal i=xz,yz,x3-y3;
def l=res(i,0); // homogeneous ideal: uses lres
l;
↦  1      3      2
↦ r <--  r <--  r
↦
↦ 0      1      2
↦
print(betti(l), "betti"); // input to betti may be of type resolution
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     2     1
↦     2:     -     1     1
↦ ------------------------
↦ total:     1     3     2
↦
l[2];         // element access may take some time
↦ _[1]=-x*gen(1)+y*gen(2)
↦ _[2]=-x2*gen(2)+y2*gen(1)+z*gen(3)
i=i,x+1;
l=res(i,0);   // inhomogeneous ideal: uses mres
l;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
ring rs=0,(x,y,z),ds;
ideal i=imap(r,i);
def l=res(i,0); // local ring not minimized: uses sres
l;
↦   1       1
↦ rs <--  rs
↦
↦ 0       1
↦ resolution not minimized yet
↦
res(i,0,0);     // local ring and minimized: uses mres
↦   1       1
↦ rs <--  rs
↦
↦ 0       1
↦
```

5.1.135 reservedName

Syntax: `reservedName ()`

Type: none

Syntax: `reservedName (` string_expression `)`

Type: int

Purpose: prints a list of all reserved identifiers (first form) or tests whether the string is a reserved identifier (second form). This includes blackbox/newstruct types.

Example:

```
reservedName();
↦ ... // output skipped
  reservedName("ring");
↦ 1
  reservedName("xyz");
↦ 0
```

See Section 5.1.103 [names], page 229; Section 4.22 [string], page 130.

5.1.136 resultant

Syntax: `resultant` (poly_expression, poly_expression, ring_variable)

Type: poly

Purpose: computes the resultant of the first and second argument with respect to the variable given as the third argument.

Example:

```
  ring r=32003,(x,y,z),dp;
  poly f=3*(x+2)^3+y;
  poly g=x+y+z;
  resultant(f,g,x);
↦ 3y3+9y2z+9yz2+3z3-18y2-36yz-18z2+35y+36z-24
```

See Section 4.17 [poly], page 120; Section 4.20 [ring], page 127.

5.1.137 ringlist

Syntax: `ringlist` (ring_expression)

Type: list

Purpose: decomposes a ring/qring into a list of 4 (or 6 in the non-commutative case, see Section 7.3.24 [ringlist (plural)], page 357) components. It is identical to `ring_list` with the exception of the first list entry.

1. the field description in the following format:

   for Q, Z/p: the characteristic, type int (0 or prime number)

   for real, complex: a list of:
   the characteristic, type int (always 0)
   the precision, type list (2 integers: external, internal precision)
   the name of the imaginary unit, type string

   for transcendental or algebraic extensions: described as a ringlist (that is, as list L with 4 entries: L[1] the characteristic, L[2] the names of the parameters, L[3] the monomial ordering for the ring of parameters (default: lp), L[4] the minimal polynomial (as ideal))

   for Z, Z/n, Z/n^m a list ["integer", [n, m]] with:
   the base n is of type int or bigint (if not given n = 0, Z/0 = Z)
   the exponent m is of type int (if not given m = 1)

2. the names of the variables (a list L of strings: L[i] is the name of the i-th variable)
3. the monomial ordering (a list L of lists): each block L[i] consists of

   the name of the ordering (string)

   parameters specifying the ordering and the size of the block (intvec : typically the weights for the variables [default: 1])
4. the quotient ideal.

From a list of such structure, a new ring may be defined by the command `ring` (see the following example). If the attribute "maxExp" of the ring is different from the default 32767, it is also set for the list.

Note: `All data which depends on a ring belong to the current ring,` not to a ring which can be constructed from a modified list. These data will be mapped via fetch to the ring to be constructed.

Example:

```
  ring r = 0,(x(1..3)),dp;
  list l = ringlist(r);
  l;
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       x(1)
↦    [2]:
↦       x(2)
↦    [3]:
↦       x(3)
↦ [3]:
↦    [1]:
↦       [1]:
↦          dp
↦       [2]:
↦          1,1,1
↦    [2]:
↦       [1]:
↦          C
↦       [2]:
↦          0
↦ [4]:
↦    _[1]=0
  //  Now change l and create a new ring, by
  //- changing the base field to the function field with parameter a,
  //- introducing one extra variable y,
  //- defining the block ordering (dp(2),wp(3,4)).
  //- define the minpoly after creating the function field
  l[1]=list(0,list("a"),list(list("lp",1)),ideal(0));
  l[2][size(l[2])+1]="y";
  l[3][3]=l[3][2]; // save the module ordering
  l[3][1]=list("dp",intvec(1,1));
  l[3][2]=list("wp",intvec(3,4));
  attrib(l,"maxExp",100); // and lower the limit for exponents to 100
  def ra = ring(l);     //creates the newring
```

```
 ra; setring ra;
↦ // coefficients: QQ(a)
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering wp
↦ //                  : names    x(3) y
↦ //                  : weights  3 4
↦ //        block   3 : ordering C
 attrib(ra,"maxExp");
↦ 65535
 list lra = ringlist(ra);
 lra[1][4]=ideal(a2+1);
 def Ra = ring(lra);
 setring Ra; Ra;
↦ // coefficients: QQ[a]/(a^2+1)
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering wp
↦ //                  : names    x(3) y
↦ //                  : weights  3 4
↦ //        block   3 : ordering C
```

See Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127; Section 5.1.138 [ring_list], page 257.

5.1.138 ring_list

Syntax: `ring_list` (ring_expression)
`ring_list` (cring_expression)

Type: list

Purpose: decomposes a ring/qring/coefficient ring into a list of 4 (or 6 in the non-commutative case, see Section 7.3.24 [ringlist (plural)], page 357) or 1/4 (for `cring`) components. It is identical to `ringlist` with the exception of the first list entry.

1. the field description as `cring`
2. the names of the variables (a list L of strings: L[i] is the name of the i-th variable)
3. the monomial ordering (a list L of lists): each block L[i] consists of

   the name of the ordering (string)

   parameters specifying the ordering and the size of the block (intvec : typically the weights for the variables [default: 1])
4. the quotient ideal.

From a list of such structure, a new ring may be defined by the command `ring` (see the following example).

Note: `All data which depends on a ring belong to the current ring,` not to a ring which can be constructed from a modified list. These data will be mapped via fetch to the ring to be constructed.

Example:

```
 ring r = 0,(x(1..3)),dp;
 list l = ring_list(r);
 ring_list(l[1]);
↦ 0
 l;
↦ [1]:
↦ QQ
↦ [2]:
↦    [1]:
↦       x(1)
↦    [2]:
↦       x(2)
↦    [3]:
↦       x(3)
↦ [3]:
↦    [1]:
↦       [1]:
↦          dp
↦       [2]:
↦          1,1,1
↦    [2]:
↦       [1]:
↦          C
↦       [2]:
↦          0
↦ [4]:
↦    _[1]=0
 //  Now change l and create a new ring, by
 //- changing the base field to ZZ/32003
 //- introducing one extra variable y,
 //- defining the block ordering (dp(2),wp(3,4)).
 //- define the minpoly after creating the function field
 l[1]=ZZ/32003;
 l[2][size(l[2])+1]="y";
 l[3][3]=l[3][2]; // save the module ordering
 l[3][1]=list("dp",intvec(1,1));
 l[3][2]=list("wp",intvec(3,4));
 def ra = ring(l);     //creates the newring
 ra; setring ra;
↦ // coefficients: ZZ/32003
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering wp
↦ //                  : names    x(3) y
↦ //                  : weights  3 4
↦ //        block   3 : ordering C
```

See Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127; Section 5.1.137 [ringlist], page 255.

5.1.139 rvar

Syntax: `rvar` (name)
`rvar` (poly_expression)
`rvar` (string_expression)

Type: int

Purpose: returns the number of the variable if the name/polynomial is a ring variable of the basering or if the string is the name of a ring variable of the basering; returns 0 if not. Hence the return value of `rvar` can also be used in a boolean context to check whether the variable exists.

Example:

```
  ring r=29,(x,y,z),lp;
  rvar(x);
↦ 1
  rvar(r);
↦ 0
  rvar(y);
↦ 2
  rvar(var(3));
↦ 3
  rvar("x");
↦ 1
```

See Section 5.1.18 [defined], page 170; Section 4.20 [ring], page 127; Section 5.1.165 [var], page 285; Section 5.1.167 [varstr], page 285.

5.1.140 sba

Syntax: `sba` (ideal_expression)
`sba` (ideal_expression`,` int_expression`,` int_expression)

Type: ideal

Purpose: returns a standard basis of an ideal with respect to the monomial ordering of the basering. A standard basis is a set of generators such that the leading terms generate the leading ideal, resp. module.
Use optional second and third arguments of type `int` to determine the respective variant of the signature-based standard basis algorithm:
The second argument specifies the internal module order `sba` uses:

`0`: induced Schreyer order on the signatures, non-incremental computation of the basis

`1`: position over term order, incremental computation of the basis

`2`: term over position order, non-incremental computation

`3`: Schreyer-weighted degree over index over leading term

The third argument specifies the rewrite order `sba` uses:

`0`: using the rewrite order described in `http://dx.doi.org/10.1016/j.jsc.2010.06.019`

`1`: using the rewrite order described in `http://dx.doi.org/10.1016/j.jsc.2011.05.004`

The standard call of `sba(i)` corresponds to `sba(i,0,1)`.

Note: The standard basis is computed with an optimized version of known signature-based algorithms like Faugere's F5 Algorithm. Whereas the correctness of the algorithms is only guaranteed for global orderings, timings for pure lexicographical orderings can be slow. In this situation you should try to compute the basis w.r.t. the graded reverse-lexicographic ordering and then convert to a basis for the lexicographical ordering using other methods (see Section 5.1.39 [fglm], page 183 and see Section D.4.10 [grwalk_lib], page 1087). If the algorithms tend to use too much memory, you should try the other implemented standard basis algorithms (see Section 5.1.151 [std], page 271, see Section 5.1.53 [groebner], page 191, and see Section 5.1.145 [slimgb], page 265).
Note that the behaviour of `sba` on an example can be rather different depending on which variant you choose (second and third argument).

Example:

```
  // incremental F5 computation
  ring r=32003,(x,y,z),dp;
  poly s1=1x2y+151xyz10+169y21;
  poly s2=1xz14+6x2y4+3z24;
  poly s3=5y10z10x+2y20z10+y10z20+11x3;
  ideal i=s1,s2,s3;
  ideal j=sba(i,1,0);
  // non-incremental F5 computation
  ring rhom=32003,(x,y,z,h),dp;
  ideal i=homog(imap(r,i),h);
  ideal j=sba(i,0,0);
  // non-incremental signature-based computation
  ring whom=32003,(x,y,z),dp;
  ideal i=fetch(r,i);
  ideal j=sba(i);
```

See Section 5.1.39 [fglm], page 183; Section 5.1.53 [groebner], page 191; Section 4.6 [ideal], page 80; Section 4.20 [ring], page 127; Section 5.1.145 [slimgb], page 265; Section 5.1.151 [std], page 271.

5.1.141 setring

Syntax: `setring` ring_name

Type: none

Purpose: changes the basering to another (already defined) ring.

Example:

```
  ring r1=0,(x,y),lp;
  // the basering is r1
  ring r2=32003,(a(1..8)),ds;
  // the basering is r2
  setring r1;
  // the basering is again r1
  nameof(basering);
↦ r1
  listvar();
↦ // r2                              [0]  ring
↦ // r1                              [0]  *ring
```

Use in procedures:

All changes of the basering by a definition of a new ring or a `setring` command in a procedure are local to this procedure. Use `keepring` to move a ring, which is local to a procedure, up by one nesting level.

See Section 5.2.11 [keepring], page 298; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127.

5.1.142 simplex

Syntax: `simplex (` matrix_expression`,` int_expression`,` int_expression`,` int_expression`,` int_expression`,` int_expression`)`

Type: list

Purpose: perform the simplex algorithm for the tableau given by the input, e.g. `simplex (`M, m, n, m1, m2, m3 `):`

M matrix of numbers :

first row describing the objective function (maximize problem), the remaining rows describing constraints;

m, n, m1, m2, m3 int :

n = number of variables; m = total number of constraints; m1 = number of inequalities `"<="` (rows 2 ... m1+1 of M); m2 = number of inequalities `">="` (rows m1+2 ... m1+m2+1 of M); m3 = number of equalities.

The following assumptions are made:

* ground field is of type `(real,N)`, N>=4;

* the matrix M is of size m x n;

* m=m1+m2+m3;

* the entries M[2,1] ,..., M[m+1,1] are non-negative;

* the variables x(i) are non-negative;

* a row b, a(1) ,..., a(n) corresponds to b+a(1)x(1)+...+a(n)x(n);

* for a <=, >=, or == constraint: add "in mind" >=0, <=0, or ==0.

The output is a list L with

* L[1] = matrix

* L[2] = int:

0 = finite solution found; 1 = unbounded; -1 = no solution; -2 = error occurred;

* L[3] = intvec :

L[3][k] = number of variable which corresponds to row k+1 of L[1];

* L[4] = intvec :

L[4][j] = number of variable which is represented by column j+1 of L[1] ("non-basis variable");

* L[5] = int :

number of constraints (= m);

* L[6] = int :

number of variables (= n).

The solution can be read off the first column of L[1] as it is done by the procedure Section D.8.4.8 [simplexOut], page 1851 in `solve.lib`.

Example:

```
   ring r = (real,10),(x),lp;

   // consider the max. problem:
   //
   //    maximize  x(1) + x(2) + 3*x(3) - 0.5*x(4)
   //
   //  with constraints:   x(1) +          2*x(3)          <= 740
   //                             2*x(2)          - 7*x(4) <=   0
   //                               x(2) -   x(3) + 2*x(4) >=   0.5
   //                      x(1) +   x(2) +   x(3) +   x(4)  =   9
   //
   matrix sm[5][5]=(  0, 1, 1, 3,-0.5,
                    740,-1, 0,-2, 0,
                      0, 0,-2, 0, 7,
                    0.5, 0,-1, 1,-2,
                      9,-1,-1,-1,-1);

   int n = 4;  // number of constraints
   int m = 4;  // number of variables
   int m1= 2;  // number of <= constraints
   int m2= 1;  // number of >= constraints
   int m3= 1;  // number of == constraints
   simplex(sm, n, m, m1, m2, m3);
↦ [1]:
↦    _[1,1]=17.025
↦    _[1,2]=-0.95
↦    _[1,3]=-0.05
↦    _[1,4]=1.95
↦    _[1,5]=-1.05
↦    _[2,1]=730.55
↦    _[2,2]=0.1
↦    _[2,3]=-0.1
↦    _[2,4]=-1.1
↦    _[2,5]=0.9
↦    _[3,1]=3.325
↦    _[3,2]=-0.35
↦    _[3,3]=-0.15
↦    _[3,4]=0.35
↦    _[3,5]=0.35
↦    _[4,1]=0.95
↦    _[4,2]=-0.1
↦    _[4,3]=0.1
↦    _[4,4]=0.1
↦    _[4,5]=0.1
↦    _[5,1]=4.725
↦    _[5,2]=-0.55
↦    _[5,3]=0.05
↦    _[5,4]=0.55
↦    _[5,5]=-0.45
↦ [2]:
↦    0
↦ [3]:
```

```
↦    5,2,4,3
↦ [4]:
↦    1,6,8,7
↦ [5]:
↦    4
↦ [6]:
↦    4
```

See Section D.8.4.8 [simplexOut], page 1851.

5.1.143 simplify

Syntax: `simplify` (poly_expression, int_expression)
`simplify` (vector_expression, int_expression)
`simplify` (ideal_expression, int_expression)
`simplify` (module_expression, int_expression)

Type: the type of the first argument

Purpose: returns the "simplified" first argument depending on the simplification rules specified by the second argument. The simplification rules are the following functions:

`1` normalize (divide by leading coefficient if this is a unit of the ground field/ring).

`2` erase zero generators/columns.

`4` erase copies of earlier listed generators/columns.

`8` erase generators/columns which a scalar multiples (w.r.t. ground field/ring) of earlier listed generators/columns.

`16` erase generators/columns whose leading monomials are copies of leading monomials of earlier listed generators/columns such that the coefficients of both leading terms are units in the ground field/ring.

`32` erase generators/columns whose leading terms are divisible by leading terms of other (not necessarily earlier) listed generators/columns.

`64` normalize each coefficient of every monomial (of every polynomial)

Example:

```
ring r=0,(x,y,z),(c,dp);
ideal i=0,2x,2x,4x,3x+y,5x2;
simplify(i,1);
↦ _[1]=0
↦ _[2]=x
↦ _[3]=x
↦ _[4]=x
↦ _[5]=x+1/3y
↦ _[6]=x2
simplify(i,2);
↦ _[1]=2x
↦ _[2]=2x
↦ _[3]=4x
↦ _[4]=3x+y
↦ _[5]=5x2
simplify(i,4);
```

```
↦ _[1]=0
↦ _[2]=2x
↦ _[3]=0
↦ _[4]=4x
↦ _[5]=3x+y
↦ _[6]=5x2
simplify(i,8);
↦ _[1]=0
↦ _[2]=2x
↦ _[3]=0
↦ _[4]=0
↦ _[5]=3x+y
↦ _[6]=5x2
simplify(i,16);
↦ _[1]=0
↦ _[2]=2x
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
↦ _[6]=5x2
simplify(i,32);
↦ _[1]=0
↦ _[2]=2x
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
↦ _[6]=0
simplify(i,32+2+1);
↦ _[1]=x
matrix A[2][3]=x,0,2x,y,0,2y;
simplify(A,2+8); // by automatic conversion to module
↦ _[1]=[x,y]
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

5.1.144 size

Syntax: `size` (string_expression)
`size` (bigint_expression)
`size` (number_expression)
`size` (intvec_expression)
`size` (bigintvec_expression)
`size` (intmat_expression)
`size` (poly_expression)
`size` (vector_expression)
`size` (ideal_expression)
`size` (module_expression)
`size` (matrix_expression)
`size` (bigintmat_expression)
`size` (list_expression)
`size` (resolution_expression)
`size` (ring_expression)

Type: int

Purpose: depends on the type of argument:

ideal or module
returns the number of (non-zero) generators.

string, intvec, bigintvec, list or resolution
returns the length, i.e., the number of characters, entries or elements.

poly or vector
returns the number of monomials.

matrix, bigintmat or intmat
returns the number of entries (rows*columns).

ring
returns the number of elements in the ground field (for Z/p and algebraic extensions) or -1

number or bigint
returns 0 for 0 or the number of words

Example:

```
  string s="hello";
  size(s);
↦ 5
  intvec iv=1,2;
  size(iv);
↦ 2
  ring r=0,(x,y,z),lp;
  poly f=x+y+z;
  size(f);
↦ 3
  vector v=[x+y,0,0,1];
  size(v);
↦ 3
  ideal i=f,y;
  size(i);
↦ 2
  module m=v,[0,1],[0,0,1],2*v;
  size(m);
↦ 4
  matrix mm[2][2];
  size(mm);
↦ 4
  ring r1=(2,a),x,dp;
  minpoly=a4+a+1;
  size(r1);
↦ 16
```

See Section 4.6 [ideal], page 80; Section 4.8 [intmat], page 90; Section 4.9 [intvec], page 93; Section 4.14 [module], page 112; Section 5.1.104 [ncols], page 231; Section 5.1.107 [nrows], page 232; Section 4.17 [poly], page 120; Section 4.22 [string], page 130; Section 4.23 [vector], page 134.

5.1.145 slimgb

Syntax:
`slimgb` (ideal_expression)
`slimgb` (module_expression)

Type: ideal or module

Purpose: Section A.2.3 [slim Groebner bases], page 714

Returns a Groebner basis of an ideal or module with respect to the monomial ordering of the basering (which has to be global).

Note: The algorithm is designed to keep polynomials slim (short with small coefficients). For details see `https://www.singular.uni-kl.de/reports/35/paper_35_full.ps.gz`. A reduced Groebner basis is returned if option(redSB) is set (see [option(redSB)], page 235). To view the progress of long running computations, use option(prot) (see [option(prot)], page 235).

Warning: Groebner basis computations with inexact coefficients can not be trusted due to rounding errors.

Example:

```
  ring r=2,(x,y,z),lp;
  poly s1=z*(x*y+1);
  poly s2=x2+x;
  poly s3=y2+y;
  ideal i=s1,s2,s3;
  slimgb(i);
↦ _[1]=y2+y
↦ _[2]=x2+x
↦ _[3]=yz+z
↦ _[4]=xz+z
```

See Section 5.1.53 [groebner], page 191; Section 4.6 [ideal], page 80; Section 5.1.111 [option], page 234; Section 4.20 [ring], page 127; Section 5.1.151 [std], page 271.

5.1.146 sortvec

Syntax: `sortvec` (ideal_expression)
`sortvec` (module_expression)

Type: intvec

Purpose: computes the permutation `v` which orders the ideal, resp. module, `I` by its initial terms, starting with the smallest, that is, `I(v[i]) < I(v[i+1])` for all `i`.

Example:

```
  ring r=0,(x,y,z),dp;
  ideal I=y,z,x,x3,xz;
  sortvec(I);
↦ 2,1,3,5,4
```

See Section D.2.3 [general_lib], page 805.

5.1.147 sqrfree

Syntax: `sqrfree` (poly_expression)
`sqrfree` (poly_expression, `0`)
`sqrfree` (poly_expression, `2`)

Type: list of ideal and intvec

Syntax: `sqrfree (` poly_expression`, 1 )`

Type: ideal

Syntax: `sqrfree (` poly_expression`, 3 )`

Type: poly

Purpose: computes the squarefree factors (as an ideal) of the polynomial together with or without the multiplicities (as an intvec) depending on the second argument:

0: returns factors and multiplicities, first factor is a constant.
May also be written with only one argument.
1: returns non-constant factors (no multiplicities).
2: returns non-constant factors and multiplicities.
3: returns the product of non-constant factors, i.e. squarefree part

Note: Not implemented for the coefficient fields real and finite fields of type `(p^n,a)`.

Example:

```
  ring r=3,(x,y,z),dp;
  poly f=(x-y)^3*(x+z)*(y-z);
  sqrfree(f);
↦ [1]:
↦    _[1]=1
↦    _[2]=-xy+xz-yz+z2
↦    _[3]=-x+y
↦ [2]:
↦    1,1,3
  sqrfree(f,1);
↦ _[1]=-xy+xz-yz+z2
↦ _[2]=-x+y
  sqrfree(f,2);
↦ [1]:
↦    _[1]=-xy+xz-yz+z2
↦    _[2]=-x+y
↦ [2]:
↦    1,3
  sqrfree(f,3);
↦ x2y-xy2-x2z-xyz-y2z-xz2+yz2
```

See Section 5.1.36 [factorize], page 180.

5.1.148 sprintf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `sprintf (` string_expression `[,` any_expressions `] )`

Return: string

Purpose: `sprintf(fmt,...);` performs output formatting. The first argument is a format control string. Additional arguments may be required, depending on the content of the control string. A series of output characters is generated as directed by the control string; these characters are returned as a string.

The control string `fmt` is simply text to be copied, except that the string may contain conversion specifications.
Type `help print;` for a listing of valid conversion specifications. As an addition to the conversions of `print`, the `%n` and `%2` conversion specification does not consume an additional argument, but simply generates a newline character.

Note: If one of the additional arguments is a list, then it should be wrapped in an additional `list()` command, since passing a list as an argument flattens the list by one level.

Example:

```
  ring r=0,(x,y,z),dp;
module m=[1,y],[0,x+z];
intmat M=betti(mres(m,0));
list l = r, m, M;
string s = sprintf("s:%s,%n l:%l", 1, 2); s;
↦ s:1,
↦  l:int(2)
s = sprintf("s:%n%s", l); s;
↦ s:
↦ (QQ),(x,y,z),(dp(3),C)
s = sprintf("s:%2%s", list(l)); s;
↦ s:
↦ (QQ),(x,y,z),(dp(3),C),y*gen(2)+gen(1),x*gen(2)+z*gen(2),1,1
s = sprintf("2l:%n%2l", list(l)); s;
↦ 2l:
↦ list("(QQ),(x,y,z),(dp(3),C)",
↦ module(y*gen(2)+gen(1),
↦ x*gen(2)+z*gen(2)),
↦ intmat(intvec(1,1 ),1,2))
↦
s = sprintf("%p", list(l)); s;
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=y*gen(2)+gen(1)
↦    _[2]=x*gen(2)+z*gen(2)
↦ [3]:
↦    1,1
s = sprintf("%;", list(l)); s;
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=y*gen(2)+gen(1)
↦    _[2]=x*gen(2)+z*gen(2)
↦ [3]:
↦    1,1
```

```
↦
s = sprintf("%b", M); s;
↦             0     1
↦  ------------------
↦      0:     1     1
↦  ------------------
↦  total:     1     1
↦
```

See also: Section 5.1.46 [fprintf], page 186; Section 5.1.120 [print], page 242; Section 5.1.121 [printf], page 244; Section 4.22 [string], page 130.

5.1.149 sres

Syntax: `sres` (ideal_expression, int_expression)
`sres` (module_expression, int_expression)

Type: resolution

Purpose: computes a free resolution of an ideal or module with Schreyer's method. The ideal, resp. module, has to be a standard basis. More precisely, let M be given by a standard basis and $A_1 = \texttt{matrix}(M)$. Then `sres` computes a free resolution of $coker(A_1) = F_0/M$

$$\ldots \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0.$$

If the int expression k is not zero then the computation stops after k steps and returns a list of modules (given by standard bases) $M_i = \texttt{module}(A_i)$, i=1..k.
`sres(M,0)` returns a list of n modules where n is the number of variables of the basering.

Even if `sres` does not compute a minimal resolution, the `betti` command gives the true betti numbers! In many cases of interest `sres` is much faster than any other known method. Let `list L=sres(M,0);` then `L[1]=M` is identical to the input, `L[2]` is a standard basis with respect to the Schreyer ordering of the first syzygy module of `L[1]`, etc. (`L`[i] $= M_i$ in the notations from above.)

Note: Accessing single elements of a resolution may require some partial computations to be finished and may therefore take some time.

Example:

```
  ring r=31991,(t,x,y,z,w),ls;
  ideal M=t2x2+tx2y+x2yz,t2y2+ty2z+y2zw,
          t2z2+tz2w+xz2w,t2w2+txw2+xyw2;
  M=std(M);
  resolution L=sres(M,0);
  L;
↦   1      35      141      209      141      43      4
↦  r <--  r <--   r <--    r <--    r <--    r <--   r
↦
↦  0      1       2        3        4        5       6
↦  resolution not minimized yet
↦
  print(betti(L),"betti");
↦             0     1     2     3     4     5
↦  ------------------------------------------
↦      0:     1     -     -     -     -     -
↦      1:     -     -     -     -     -     -
↦     2:     -     -     -     -     -     -
↦     3:     -     4     -     -     -     -
↦     4:     -     -     -     -     -     -
↦     5:     -     -     -     -     -     -
↦     6:     -     -     6     -     -     -
↦     7:     -     -     9    16     2     -
↦     8:     -     -     -     2     5     1
↦ ------------------------------------------
↦ total:     1     4    15    18     7     1
↦
```

See Section 5.1.4 [betti], page 159; Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 4.7 [int], page 85; Section 5.1.83 [lres], page 215; Section 5.1.93 [minres], page 223; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section 5.1.134 [res], page 253; Section 5.1.156 [syz], page 280.

5.1.150 status

Syntax: `status` (link_expression, string_expression)

Type: string

Syntax: `status` (link_expression, string_expression, string_expression)

Type: int

Purpose: returns the status of the link as asked for by the second argument. If a third argument is given, the result of the comparison to the status string is returned: `(status(l,s1)==s2)` is equivalent to `status(l,s1,s2)`.
The following string expressions are allowed:

`"name"` the name string given by the definition of the link (usually the filename)

`"type"` returns `"ASCII"`, `"DBM"` or `"ssi"`

`"open"` returns `"yes"` or `"no"`

`"openread"`
returns `"yes"` or `"no"`

`"openwrite"`
returns `"yes"` or `"no"`

`"read"` returns `"ready"` or `"not ready"`

`"write"` returns `"ready"` or `"not ready"`

`"mode"` returns (depending on the type of the link and its status) `"","w","a","r"` or `"rw"`

`"exists"` returns `"yes"` or `"no"`: existence of the filename for ASCII/ssi links

Syntax: `status` (list_expression, int_expression)

Type: int

Purpose: the list should be a list L of links, the second argument a timeout in 1/10 seconds. Returns

`-2` select returns an error

`-1` all links are closed/at eof

`0` timeout

`>0` (at least) L[i] is ready

Example:

```
  link l=":w example.txt";
  status(l,"write");
↦ not ready
  open(l);
  status(l,"write","ready");
↦ 1
  close(l);
```

See Section 4.10 [link], page 96; Section 5.1.110 [open], page 234; Section 5.1.130 [read], page 250; Section 5.1.174 [write], page 289.

5.1.151 std

Syntax:
`std` (ideal_expression)
`std` (module_expression)
`std` (smatrix_expression)
`std` (ideal_expression`,` bigintvec_expression)
`std` (module_expression`,` bigintvec_expression)
`std` (ideal_expression`,` intvec_expression`,` intvec_expression)
`std` (module_expression`,` intvec_expression`,` intvec_expression)
`std` (ideal_expression`,` poly_expression)
`std` (module_expression`,` vector_expression)
`std` (ideal_expression`,` ideal_expression)
`std` (module_expression`,` module_expression)
`std` (ideal_expression`,` poly_expression`,` intvec_expression`,` intvec_expression)
`std` (module_expression`,` poly_expression`,` intvec_expression`,` intvec_expression)

Type: ideal, module or smatrix

Purpose: returns a standard basis of an ideal or module with respect to the monomial ordering of the basering. For Letterplace rings, a twosided Groebner basis is computed. A standard basis is a set of generators such that the leading terms generate the leading ideal, resp. module.
Use an optional second argument of type intvec as Hilbert series (result of `hilb(i,1)`, see Section 5.1.56 [hilb], page 195), if the ideal, resp. module, is homogeneous (Hilbert driven standard basis computation, Section 5.1.153 [stdhilb], page 273). If the ideal is quasihomogeneous with some weights w and if the Hilbert series is computed w.r.t. to these weights, then use w as third argument.
Use an optional second argument of type poly/vector/ideal, resp. module, to construct the standard basis from an already computed one (given as the first argument) and additional generator(s) (the second argument).
4 arguments `G,p,hv,w` are the combination of the above: standard basis `G`, additional generator `p`, hilbert function `hv` w.r.t. weights `w`.

Warning: Groebner basis computations with inexact coefficients can not be trusted due to rounding errors.

Note: The standard basis is computed with a (more or less) straight-forward implementation of the classical Buchberger (resp. Mora) algorithm. For global orderings, use the

groebner command instead (see Section 5.1.53 [groebner], page 191), which heuristically chooses the "best" algorithm to compute a Groebner basis.
To view the progress of long running computations, use option(prot) (see [option(prot)], page 235).

Example:

```
  // local computation
  ring r=32003,(x,y,z),ds;
  poly s1=1x2y+151xyz10+169y21;
  poly s2=1xz14+6x2y4+3z24;
  poly s3=5y10z10x+2y20z10+y10z20+11x3;
  ideal i=s1,s2,s3;
  ideal j=std(i);
  degree(j);
↦ // dimension (local)   = 0
↦ // multiplicity = 1512
  // Hilbert driven elimination (standard)
  ring rhom=32003,(x,y,z,h),dp;
  ideal i=homog(imap(r,i),h);
  ideal j=std(i);
  intvec iv=hilb(j,1);
  ring rlex=32003,(x,y,z,h),lp;
  ideal i=fetch(rhom,i);
  ideal j=std(i,iv);
  j=subst(j,h,1);
  j[1];
↦ z64
  // Hilbert driven elimination (ideal is quasihomogeneous)
  intvec w=10,1,1;
  ring whom=32003,(x,y,z),wp(w);
  ideal i=fetch(r,i);
  ideal j=std(i);
  intvec iw=hilb(j,1,w);
  ring wlex=32003,(x,y,z),lp;
  ideal i=fetch(whom,i);
  ideal j=std(i,iw,w);
  j[1];
↦ z64
```

See Section 5.1.34 [facstd], page 179; Section 5.1.39 [fglm], page 183; Section 5.1.53 [groebner], page 191; Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.100 [mstd], page 227; Section 5.1.111 [option], page 234; Section 4.20 [ring], page 127; Section 4.21 [smatrix], page 129; Section 5.1.152 [stdfglm], page 272; Section 5.1.153 [stdhilb], page 273.

5.1.152 stdfglm

Procedure from library standard.lib (see Section D.1 [standard_lib], page 793).

Syntax: stdfglm (ideal_expression)
stdfglm (ideal_expression, string_expression)

Type: ideal

Purpose: computes the standard basis of the ideal in the basering via fglm from the ordering given as the second argument to the ordering of the basering. If no second argument is given, "dp" is used. The standard basis for the given ordering (resp. for "dp") is computed via the command groebner except if a further argument "std" or "slimgb" is given in which case std resp. slimgb is used.

Example:

```
   ring r  = 0,(x,y,z),lp;
ideal i = y3+x2,x2y+x2,x3-x2,z4-x2-y;
stdfglm(i);                   //uses fglm from "dp" (with groebner) to "lp"
↦ _[1]=z12
↦ _[2]=yz4-z8
↦ _[3]=y2+y-z8-z4
↦ _[4]=xy-xz4-y+z4
↦ _[5]=x2+y-z4
stdfglm(i,"std");             //uses fglm from "dp" (with std) to "lp"
↦ _[1]=z12
↦ _[2]=yz4-z8
↦ _[3]=y2+y-z8-z4
↦ _[4]=xy-xz4-y+z4
↦ _[5]=x2+y-z4
ring s  = (0,x),(y,z,u,v),lp;
minpoly = x2+1;
ideal i = u5-v4,zv-u2,zu3-v3,z2u-v2,z3-uv,yv-zu,yu-z2,yz-v,y2-u,u-xy2;
weight(i);
↦ 2,3,4,5
stdfglm(i,"(a(2,3,4,5),dp)"); //uses fglm from "(a(2,3,4,5),dp)" to "lp"
↦ _[1]=v2
↦ _[2]=u
↦ _[3]=zv
↦ _[4]=z2
↦ _[5]=yv
↦ _[6]=yz-v
↦ _[7]=y2
```

See also: Section 5.1.39 [fglm], page 183; Section 5.1.53 [groebner], page 191; Section 5.1.145 [slimgb], page 265; Section 5.1.151 [std], page 271; Section 5.1.153 [stdhilb], page 273.

5.1.153 stdhilb

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `stdhilb` (ideal_expression)
`stdhilb` (module_expression)
`stdhilb` (ideal_expression, intvec_expression)
`stdhilb` (module_expression, intvec_expression)
`stdhilb` (ideal_expression`,` list of string_expressions, and intvec_expression)

Type: type of the first argument

Purpose: Compute a Groebner basis of the ideal/module in the basering by using the Hilbert driven Groebner basis algorithm. If an argument of type string, stating `"std"` resp. `"slimgb"`, is given, the standard basis computation uses `std` or `slimgb`, otherwise a heuristically chosen method (default)
If an optional second argument w of type intvec is given, w is used as variable weights.

If w is not given, it is computed as w[i] = deg(var(i)). If the ideal is homogeneous w.r.t. w then the Hilbert series is computed w.r.t. to these weights.

Theory: If the ideal is not homogeneous compute first a Groebner basis of the homogenization [w.r.t. the weights w] of the ideal/module, then the Hilbert function and, finally, a Groebner basis in the original ring by using the computed Hilbert function. If the given w does not coincide with the variable weights of the basering, the result may not be a groebner basis in the original ring.

Note: 'Homogeneous' means weighted homogeneous with respect to the weights w[i] of the variables var(i) of the basering. Parameters are not converted to variables.

Example:

```
   ring  r = 0,(x,y,z),lp;
ideal i = y3+x2,x2y+x2z2,x3-z9,z4-y2-xz;
ideal j = stdhilb(i); j;
↦ j[1]=z10
↦ j[2]=yz9
↦ j[3]=2y2z4-z8
↦ j[4]=2y3z3-2y2z5-yz7
↦ j[5]=y4+y3z2
↦ j[6]=xz+y2-z4
↦ j[7]=xy2-xz4-y3z
↦ j[8]=x2+y3
ring  r1 = 0,(x,y,z),wp(3,2,1);
ideal  i = y3+x2,x2y+x2z2,x3-z9,z4-y2-xz;  //ideal is homogeneous
ideal j = stdhilb(i,"std"); j;
↦ j[1]=y2+xz-z4
↦ j[2]=x2-xyz+yz4
↦ j[3]=2xz5-z8
↦ j[4]=2xyz4-yz7+z9
↦ j[5]=z10
↦ j[6]=2yz9+z11
//this is equivalent to:
intvec v = hilb(std(i),1);
ideal j1 = std(i,v,intvec(3,2,1)); j1;
↦ j1[1]=y2+xz-z4
↦ j1[2]=x2-xyz+yz4
↦ j1[3]=2xz5-z8
↦ j1[4]=2xyz4-yz7+z9
↦ j1[5]=z10
↦ j1[6]=yz9
size(NF(j,j1))+size(NF(j1,j));            //j and j1 define the same ideal
↦ 0
```

See also: Section 5.1.53 [groebner], page 191; Section 5.1.145 [slimgb], page 265; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272.

5.1.154 subst

Syntax: `subst` (poly_expression, variable, poly_expression)
`subst` (poly_expression, variable, poly_expression , ... variable, poly_expression)
`subst` (vector_expression, variable, poly_expression)
`subst` (ideal_expression, variable, poly_expression)
`subst` (module_expression, variable, poly_expression)

Type: poly, vector, ideal or module (corresponding to the first argument)

Purpose: substitutes one or more ring variable(s)/parameter variable(s) by (a) polynomial(s). Note that in the case of more than one substitution pair, the substitutions will be performed sequentially and not simultaneously. The below examples illustrate this behaviour.
Note, that the coefficients must be polynomial when substituting a parameter.

Example:

```
  ring r=0,(x,y,z),dp;
  poly f=x2+y2+z2+x+y+z;
  subst(f,x,y,y,z);   // first substitute x by y, then y by z
↦ 3z2+3z
  subst(f,y,z,x,y);   // first substitute y by z, then x by y
↦ y2+2z2+y+2z
```

See Section 4.6 [ideal], page 80; Section 4.12 [map], page 106; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section D.2.8.17 [substitute], page 895; Section 4.23 [vector], page 134.

5.1.155 system

Syntax: `system` (string_expression)
`system` (string_expression, expression)

Type: depends on the desired function, may be none

Purpose: interface to internal data and the operating system. The string_expression determines the command to execute. Some commands require an additional argument (second form) where the type of the argument depends on the command. See below for a list of all possible commands.

Note: Not all functions work on every platform.

Functions:

`system("alarm"`, int)
abort the Singular process after computing for that many seconds (system+user cpu time).

`system("absFact"`, poly)
absolute factorization of the polynomial (from a polynomial ring over a transzedental extension) Returns a list of the ideal of the factors, intvec of multiplicities, ideal of minimal polynomials and the number of factors.

`system("blackbox")`
list all blackbox data types.

`system("browsers");`
returns a string about available help browsers. See Section 3.1.3 [The online help system], page 15.

`system("bracket"`, poly, poly)
returns the Lie bracket [p,q].

`system("complexNearZero"`, number_expression)
checks for a small value for floating point numbers

`system("contributors")`
returns names of people who contributed to the Singular kernel as string.

`system("content",p)`
: returns p/content(p) for poly/vector

`system("cpu")`
: returns the number of cpus as int (for creating multiple threads/processes). (see `system("--cpus")`).

`system("denom_list")`
: returns the list of denominators (number) which occurred in the latest std computationi(s). Is reset to the empty list at ring changes or by this system call.

`system("eigenvals",` matrix `)`
: returns the list of the eigenvalues of the matrix (as ideal, intvec). (see `system("hessenberg")`).

`system("env",` ring `)`
: returns the enveloping algebra (i.e. R tensor R^opp) See `system("opp")`.

`system("executable",` string `)`
: returns the path of the command given as argument or the empty string (for: not found) See `system("Singular")`. See `system("getenv","PATH")`.

`system("getenv",` string_expression)
: returns the value of the shell environment variable given as the second argument. The return type is string.

`system("getPrecDigits")`
: returns the precision for floating point numbers

`system("gmsnf",` ideal, ideal, matrix,int, int `)`
: Gauss-Manin system: for gmspoly.lib, gmssing.lib

`system("HC")`
: returns the degree of the "highest corner" from the last std computation (or 0).

`system("hessenberg",` matrix `)`
: returns the Hessenberg matrix (via QR algorithm).

`system("install",` s1, s2, p3, i4 `)`
: install a new method p3 for s2 for the newstruct type s1. s2 must be a reserved operator with i4 operands (i4 may be 1,2,3; use 4 for more than 3 or a varying number of arguments) See See Section 4.24.4 [Commands for user defined types], page 138.

`system("LLL",` B `)`
: B must be a matrix or an intmat. Interface to NTLs LLL (Exact Arithmetic Variant over ZZ). Returns the same type as the input.
B is an m x n matrix, viewed as m rows of n-vectors. m may be less than, equal to, or greater than n, and the rows need not be linearly independent. B is transformed into an LLL-reduced basis. The first m-rank(B) rows of B are zero.
More specifically, elementary row transformations are performed on B so that the non-zero rows of new-B form an LLL-reduced basis for the lattice spanned by the rows of old-B.

`system("nblocks")` or `system("nblocks",` ring_name `)`

returns the number of blocks of the given ring, or of the current basering, if no second argument is given. The return type is int.

`system("nc_hilb",` ideal, int, [,...] `)`

internal support for ncHilb.lib, return nothing

`system("neworder",` ideal `)`

string of the ring variables in an heurically good order for `char_series`

`system("newstruct")`

list all newstruct data types.

`system("opp",` ring `)`

returns the opposite ring.

`system("oppose",` ring R, poly p `)`

returns the opposite polynomial of p from R.

`system("pcvLAddL",` list, list `)`

`system("pcvPMulL",` poly, list `)`
`system("pcvMinDeg",` poly `)`
`system("pcvP2CV",` list, int, int `)`
`system("pcvCV2P",` list, int, int `)`
`system("pcvDim",` int, int `)`
`system("pcvBasis",` int, int `)` internal for mondromy.lib

`system("pid")`

returns the process number as int (for creating unique names).

`system("random")` or `system("random",` int `)`

returns or sets the seed of the random generator.

`system("reduce_bound",` poly, ideal, int `)`

or `system("reduce_bound",` ideal, ideal, int `)`
or `system("reduce_bound",` vector, module, int `)`
or `system("reduce_bound",` module, module, int `)` returns the normalform of the first argument wrt. the second up to the given degree bound (wrt. total degree)

`system("reserve",` int `)`

reserve a port and listen with the given backlog. (see `system("reservedLink")`).

`system("reservedLink")`

accept a connect at the reserved port and return a (write-only) link to it. (see `system("reserve")`).

`system("rref",` matrix `)`

return a reduced row echelon form of the constant matrix M (see `system("rref")`).

`system("semaphore",` string, int `)`

operations for semaphores: string may be `"init"`, `"exists"`, `"acquire"`, `"try_acquire"`, `"release"`, `"get_value"`, and int is the number of the semaphore. Returns -2 for wrong command, -1 for error or the result of the command.

`system("semic", list, list )`
: or `system("semic", list, list, int )` computes from list of spectrum numbers and list of spectrum numbers the semicontinuity index (qh, if 3rd argument is 1).

`system("setenv",string_expression, string_expression)`
: sets the shell environment variable given as the second argument to the value given as the third argument. Returns the third argument. Might not be available on all platforms.

`system("sh", string_expression )`
: shell escape, returns the return code of the shell as int. The string is sent literally to the shell.

`system("shrinktest", poly, i2 )`
: internal for shift algebra (with i2 variables): shrink the poly

`system("Singular")`
: returns the absolute (path) name of the running SINGULAR as string.

`system("SingularBin")`
: returns the absolute path name of directory of the running SINGULAR as string (ending in /)

`system("SingularLib")`
: returns the colon separated library search path name as string.

`system("spadd", list, list )`
: or `system("spadd", list, list, int )` computes from list of spectrum numbers and list of spectrum numbers the sum of the lists.

`system("spectrum", poly )`
: or `system("spectrum", poly, int )`

`system("spmul", list, int )`
: or `system("spmul", list, list, int )` computes from list of spectrum numbers the multiple of it.

`system("std_syz", module, int )`
: compute a partial groebner base of a module, stop after the given column

`system("tensorModuleMult", int, module )`
: internal for sheafcoh.lib (see id_TensorModuleMult)

`system("twostd", ideal )`
: returns the two-sided standard basis of the two-sided ideal.

`system("uname")`
: returns a string identifying the architecture for which SINGULAR was compiled.

`system("verifyGB", ideal_expression/module_expression )`
: checks, if an ideal/module is a Groebner base

`system("version")`
: returns the version number of SINGULAR as int. (Version a-b-c-d returns a*1000+b*100+c*10+d)

`system("with")`
: without an argument: returns a string describing the current version of Singular, its build options, the used path names and other configurations with a string argument: test for that feature and return an int.

`system("--cpus")`
: returns the number of available cpu cores as int (for using multiple cores). (see `system("cpu")`).

`system("–")`
: prints the values of all options.

`system("–long_option_name")`
: returns the value of the (command-line) option long_option_name. The type of the returned value is either string or int. See Section 3.1.6 [Command line options], page 19, for more info.

`system("–long_option_name", expression)`
: sets the value of the (command-line) option long_option_name to the value given by the expression. Type of the expression must be string, or int. See Section 3.1.6 [Command line options], page 19, for more info. Among others, this can be used for setting the seed of the random number generator, the used help browser, the minimal display time, or the timer resolution.

Example:

```
// a listing of the current directory:
system("sh","ls");
// execute a shell, return to SINGULAR with exit:
system("sh","sh");
string unique_name="/tmp/xx"+string(system("pid"));
unique_name;
↦ /tmp/xx4711
system("uname")
↦ ix86-Linux
system("getenv","PATH");
↦ /bin:/usr/bin:/usr/local/bin
system("Singular");
↦ /usr/local/bin/Singular
// report value of all options
system("--");
↦ // --batch           0
↦ // --execute
↦ // --sdb             0
↦ // --echo            1
↦ // --profile         0
↦ // --quiet           1
↦ // --sort            0
↦ // --random          12345678
↦ // --no-tty          1
↦ // --user-option
↦ // --allow-net       0
↦ // --browser
↦ // --cntrlc
↦ // --emacs           0
↦ // --log
↦ // --no-stdlib       0
↦ // --no-rc           1
↦ // --no-warn         0
↦ // --no-out          0
↦ // --no-shell        0
↦ // --min-time        "0.5"
↦ // --cpus            4
↦ // --threads         4
↦ // --flint-threads   1
↦ // --MPport
↦ // --MPhost
↦ // --link
↦ // --ticks-per-sec   1
// set minimal display time to 0.02 seconds
system("--min-time", "0.02");
// set timer resolution to 0.01 seconds
system("--ticks-per-sec", 100);
// re-seed random number generator
system("--random", 12345678);
// allow your web browser to access HTML pages from the net
system("--allow-net", 1);
// and set help browser to firefox
system("--browser", "firefox");
↦ // ** No help browser 'firefox' available.
↦ // ** Setting help browser to 'dummy'.
```

5.1.156 syz

Syntax: syz (ideal_expression)
syz (module_expression)
syz (ideal_expression, string_expression)
syz (module_expression, string_expression)

Type: module

Purpose: computes the first syzygy (i.e., the module of relations of the given generators) of the ideal, resp. module.
An optional second argument specifies the Groebner base algorithm to use. Possible values are "std"(default) and "slimgb".
Only for use of "std": If option(returnSB) is set, a standard basis is returned, otherwise a generating set.

Example:

```
  ring R=0,(x,y),(c,dp);
  ideal i=x,y;
  module s=syz(i);
  s;
↦ s[1]=[y,-x]
  matrix(i)*matrix(s);
↦ _[1,1]=0
  s=syz(i,"slimgb");
  s;
↦ s[1]=[y,-x]
```

See Section 5.1.48 [fres], page 188; Section 5.1.58 [hres], page 197; Section 4.6 [ideal], page 80; Section 5.1.80 [lift], page 211; Section 5.1.81 [liftstd], page 212; Section 5.1.83 [lres], page 215; Section 4.14 [module], page 112; Section 5.1.98 [mres], page 225; Section D.4.21 [nfmodsyz_lib], page 1177; Section 5.1.106 [nres], page 232; Section 5.1.111 [option], page 234; Section 5.1.134 [res], page 253; Section 5.1.149 [sres], page 269.

5.1.157 tensor

Syntax: `tensor` (matrix_expression , matrix_expression)
`tensor` (module_expression , module_expression)
`tensor` (smatrix_expression , smatrix_expression)

Type: same as the first argument

Purpose: computes the tensor product (Kronecker product) of A and B

Example:

```
  ring r=32003,(x,y,z),(c,ds);
  matrix A[3][3]=1,2,3,4,5,6,7,8,9;
  matrix B[2][2]=x,y,0,z;
  print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
  print(B);
↦ x,y,
↦ 0,z
  print(tensor(A,B));
↦ x, y, 2x,2y,3x,3y,
↦ 0, z, 0, 2z,0, 3z,
↦ 4x,4y,5x,5y,6x,6y,
↦ 0, 4z,0, 5z,0, 6z,
↦ 7x,7y,8x,8y,9x,9y,
↦ 0, 7z,0, 8z,0, 9z
```

See Section 4.13 [matrix], page 108; Section 4.14 [module], page 112.

5.1.158 trace

Syntax: `trace` (intmat_expression)
`trace` (matrix_expression)

Type: int, if the argument is an intmat, resp.
poly, if the argument is a matrix

Purpose: returns the trace of an intmat, resp. matrix.

Example:

```
  intmat m[2][2]=1,2,3,4;
  print(m);
↦      1     2
↦      3     4
  trace(m);
↦ 5
```

See Section 4.8 [intmat], page 90; Section 4.13 [matrix], page 108.

5.1.159 transpose

Syntax: `transpose` (intmat_expression)
`transpose` (matrix_expression)
`transpose` (smatrix_expression)
`transpose` (module_expression)

Type: intmat, matrix, or module, corresponding to the argument

Purpose: transposes a matrix.

Example:

```
  ring R=0,x,dp;
  matrix m[2][3]=1,2,3,4,5,6;
  print(m);
↦ 1,2,3,
↦ 4,5,6
  print(transpose(m));
↦ 1,4,
↦ 2,5,
↦ 3,6
```

See Section 4.8 [intmat], page 90; Section 4.13 [matrix], page 108; Section 4.14 [module], page 112; Section 4.21 [smatrix], page 129.

5.1.160 type

Syntax: `type` name ;
`type` (name) ;

Type: none

Purpose: prints the name, level, type and value of a variable. To display the value of an expression, it is sufficient to type the expression followed by `;`.

Example:

```
  int i=3;
  i;
↦ 3
  type(i);
↦ // i int 3
```

See Chapter 4 [Data types], page 73; Section 5.1.82 [listvar], page 213; Section 5.1.120 [print], page 242.

5.1.161 typeof

Syntax: `typeof` (expression)

Type: string

Purpose: returns the type of an expression as string.

Returns the type of the first list element if the expression is an expression list.

Possible types are: `"ideal"`, `"int"`, `"intmat"`, `"intvec"`, `"list"`, `"map"`, `"matrix"`, `"module"`, `"number"`, `"none"`, `"poly"`, `"proc"`, `"qring"`, `"resolution"`, `"ring"`, `"string"`, `"vector"`.

For internal use only is the type `"?unknown type?"`.

Example:

```
  int i=9; i;
↦ 9
  typeof(_);
↦ int
  print(i);
↦ 9
  typeof(_);
↦ string
  type i;
↦ // i int 9
  typeof(_);
↦ string
  string s=typeof(i);
  s;
↦ int
  typeof(s);
↦ string
  proc p() {  "hello"; return();}
  p();
↦ hello
  typeof(_);
↦ ?unknown type?
```

See Chapter 4 [Data types], page 73; Section 5.1.160 [type], page 282.

5.1.162 univariate

Syntax: `univariate` (poly_expression)

Type: int

Purpose: returns 0 for not univariate, -1 for a constant or the number of the variable of the univariate polynomial.

Example:

```
  ring r=0,(x,y,z),dp;
  univariate(x2+1);
↦ 1
  univariate(x2+y+1);
↦ 0
  univariate(1);
↦ -1
  univariate(var(2));
↦ 2
  var(univariate(z));
↦ z
```

See Section 5.1.77 [leadexp], page 210; Section 5.1.165 [var], page 285.

5.1.163 uressolve

Syntax: `uressolve` (ideal_expression, int_expression, int_expression, int_expression)

Type: list

Purpose: computes all complex roots of a zerodimensional ideal.
Makes either use of the multipolynomial resultant of Macaulay (second argument = 1), which works only for homogeneous ideals, or uses the sparse resultant of Gelfand, Kapranov and Zelevinsky (second argument = 0).
The sparse resultant algorithm uses a mixed polyhedral subdivision of the Minkowski sum of the Newton polytopes in order to construct the sparse resultant matrix. Its determinant is a nonzero multiple of the sparse resultant. The u-resultant of B.\ L. van der Waerden and Laguerre's algorithm are used to determine the complex roots.
The third argument defines the precision of the fractional part if the ground field is the field of rational numbers, otherwise it will be ignored.
The fourth argument (can be 0, 1 or 2) gives the number of extra runs of Laguerre's algorithm (with corrupted roots), leading to better results.

Note: If the ground field is the field of complex numbers, the elements of the list are of type number, otherwise of type string.

See Section 5.1.74 [laguerre], page 208; Section 5.1.97 [mpresmat], page 225.

5.1.164 vandermonde

Syntax: `vandermonde (` ideal_expression`,` ideal_expression`,` int_expression `)`

Type: poly

Purpose: `vandermonde(p,v,d)` computes the (unique) polynomial of degree `d` with prescribed values `v[1],...,v[N]` at the points $\mathtt{p}^0, \ldots, \mathtt{p}^{N-1}$, `N=(d+1)`^{n}, n the number of ring variables.

The returned polynomial is $\sum c_{\alpha_1 \ldots \alpha_n} \cdot x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$, where the coefficients $c_{\alpha_1 \ldots \alpha_n}$ are the solution of the (transposed) Vandermonde system of linear equations

$$\sum_{\alpha_1+\ldots+\alpha_n \le d} c_{\alpha_1 \ldots \alpha_n} \cdot \mathtt{p}_1^{(k-1)\alpha_1} \cdot \ldots \cdot \mathtt{p}_n^{(k-1)\alpha_n} = \mathtt{v}[k], \quad k = 1, \ldots, \mathtt{N}.$$

Note: the ground field has to be the field of rational numbers. Moreover, `ncols(p)==`n, the number of variables in the basering, and all the given generators have to be numbers different from 0,1 or -1. Finally, `ncols(v)==(d+1)`^{n}, and all given generators have to be numbers.

Example:

```
ring r=0,(x,y),dp;
// determine f with deg(f)=2 and with given values v of f
// at 9 points: (2,3)^0=(1,1),...,(2,3)^8=(2^8,3^8)
// valuation point: (2,3)
ideal p=2,3;
ideal v=1,2,3,4,5,6,7,8,9;
poly ip=vandermonde(p,v,2);
ip[1..5];  //  the 5 first terms of ip:
↦ -1/9797760x2y2-595/85536x2y+55/396576xy2+935/384x2-1309/3240xy
// compute value of ip at the point 2^8,3^8, result must be 9
subst(subst(ip,x,2^8),y,3^8);
↦ 9
```

5.1.165 var

Syntax: `var` (int_expression)

Type: poly

Purpose: `var(n)` returns the n-th ring variable.

Example:

```
  ring r=0,(x,y,z),dp;
  var(2);
↦ y
```

See Section 4.7 [int], page 85; Section 5.1.109 [nvars], page 233; Section 4.20 [ring], page 127; Section 5.1.139 [rvar], page 258; Section 5.1.162 [univariate], page 283; Section 5.1.167 [varstr], page 285.

5.1.166 variables

Syntax: `variables` (poly_expression)
`variables` (ideal_expression)
`variables` (matrix_expression)

Type: ideal

Purpose: `variables(p)` returns the list of all ring variables the argument depends on.

Example:

```
  ring r=0,(x,y,z),dp;
  variables(2);
↦ _[1]=0
  variables(x+y2);
↦ _[1]=x
↦ _[2]=y
  variables(ideal(x+y2,x3y,z));
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
  string(variables(ideal(x+y2,x3y,z)));
↦ x,y,z
```

See Section 5.1.77 [leadexp], page 210; Section 5.1.109 [nvars], page 233; Section 5.1.162 [univariate], page 283; Section 5.1.165 [var], page 285; Section 5.1.167 [varstr], page 285.

5.1.167 varstr

Syntax: `varstr` (ring_name)
`varstr` (int_expression)
`varstr` (ring_name, int_expression)

Type: string

Purpose: returns the list of the names of the ring variables as a string or the name of the n-th ring variable, where n is given by the int_expression.
If the ring name is omitted, the basering is used, thus `varstr(n)` is equivalent to `varstr(basering,n)`.

Example:

```
    ring r=0,(x,y,z),dp;
    varstr(r);
  ↦ x,y,z
    varstr(r,1);
  ↦ x
    varstr(2);
  ↦ y
```

See Section 5.1.7 [charstr], page 162; Section 4.7 [int], page 85; Section 5.1.109 [nvars], page 233; Section 5.1.113 [ordstr], page 239; Section 5.1.116 [parstr], page 240; Section 4.20 [ring], page 127; Section 5.1.165 [var], page 285.

5.1.168 vdim

Syntax: `vdim` (ideal_expression)
`vdim` (module_expression)

Type: int

Purpose: computes the vector space dimension of the ring, resp. free module, modulo the ideal, resp. module, generated by the initial terms of the given generators. If the generators form a standard basis, this is the same as the vector space dimension of the ring, resp. free module, modulo the ideal, resp. module.
If the ideal, resp. module, is not zero-dimensional, -1 is returned.

Example:

```
    ring r=0,(x,y),ds;
    ideal i=x2+y2,x2-y2;
    ideal j=std(i);
    vdim(j);
  ↦ 4
```

See Section D.6.20.1 [codim], page 1756; Section 5.1.20 [degree], page 171; Section 5.1.25 [dim], page 174; Section 4.6 [ideal], page 80; Section 5.1.69 [kbase], page 205; Section 5.1.101 [mult], page 228; Section 5.1.151 [std], page 271.

5.1.169 waitall

Syntax: `waitall` (list_expression)
`waitall` (list_expression `,` int_expression)

Type: int

Purpose: Expects a list of open links (of mode ssi:fork, ssi:tcp) and waits until all of them are finished, i.e., are ready to be read.
In the first case, the command waits for all links to finish (or to crash, see below) and may therefore run forever.
In the second case, a timeout in milliseconds can be provided, forcing the command to terminate after the specified time. If the given timeout is 0, the command checks whether all links are finished or not, but does not wait for any link (polling).
Return values are:
-1: The read state of all links is `"eof"`, see Section 4.10 [link], page 96, Section 5.1.150 [status], page 270. This might happen if all the links crashed.
0: timeout (or polling): None of the links is ready.

1: All links are ready. (Note: There might be links whose read state is "eof", but at least one link is ready.)

Example:

```
      link l1 = "ssi:fork"; open(l1);
      link l2 = "ssi:fork"; open(l2);
      link l3 = "ssi:fork"; open(l3);
      list l = list(l1,l2,l3);
      write(l1, quote(system("sh", "sleep 15")));
      write(l2, quote(system("sh", "sleep 10")));
      write(l3, quote(system("sh", "sleep 11")));
      waitall(l, 5000); // terminates after 5sec with result 0
    ↦ 0
      waitall(l);       // terminates after 10 more sec
    ↦ 1
      close(l1);
      close(l2);
      close(l3);
```

See Section 5.1.170 [waitfirst], page 287.

5.1.170 waitfirst

Syntax: `waitfirst` (list_expression)
`waitfirst` (list_expression `,` int_expression)

Type: int

Purpose: Expects a list of open links (of mode ssi:fork, ssi:tcp) and waits until the first of them is finished, i.e., is ready to be read.
In the first case, the command waits until the first link is finished (or all of them crashed, see below) and may therefore run forever.
In the second case, a timeout in milliseconds can be provided, forcing the command to terminate after the specified time. If the given timeout is 0, the command checks whether one of the links is finished or not, but does not wait for any link (polling).
Return values are:
-1: The read state of all links is "eof", see Section 4.10 [link], page 96, Section 5.1.150 [status], page 270. This might happen if all the links crashed.
0: timeout (or polling): None of the links is ready.
i>1: At least the link whose list index is i is ready.

Example:

```
      link l1 = "ssi:fork"; open(l1);
      link l2 = "ssi:fork"; open(l2);
      link l3 = "ssi:fork"; open(l3);
      list l = list(l1,l2,l3);
      write(l1, quote(system("sh", "sleep 15")));
      write(l2, quote(system("sh", "sleep 13")));
      write(l3, quote(system("sh", "sleep 11")));
      waitfirst(l, 5000); // terminates after 5sec with result 0
    ↦ 0
      waitfirst(l);       // terminates after 6 more sec with result 3
    ↦ 3
      close(l1);
```

```
  close(l2);
  close(l3);
```

See Section 5.1.169 [waitall], page 286.

5.1.171 wedge

Syntax: `wedge` (matrix_expression, int_expression)

Type: matrix

Purpose: `wedge(M,n)` computes the `n`-th exterior power of the matrix `M`.

Example:

```
  ring r;
  matrix m[2][3]=x,y,y,z,z,x;
  print(m);
↦ x,y,y,
↦ z,z,x
  print(wedge(m,2));
↦ xz-yz,-x2+yz,xy-yz
```

See Section 4.7 [int], page 85; Section 4.13 [matrix], page 108; Section 5.1.92 [minor], page 221.

5.1.172 weight

Syntax: `weight` (ideal_expression)
`weight` (module_expression)

Type: intvec

Purpose: computes an "optimal" weight vector for an ideal, resp. module, which may be used as weight vector for the variables in order to speed up the standard basis algorithm. If the input is weighted homogeneous, a weight vector for which the input is weighted homogeneous is found.

Example:

```
  ring h1=32003,(t,x,y,z),dp;
  ideal i=
  9x8+y7t3z4+5x4y2t2+2xy2z3t2,
  9y8+7xy6t+2x5y4t2+2x2yz3t2,
  9z8+3x2y3z2t4;
  intvec e=weight(i);
  e;
↦ 5,7,5,7
  ring r=32003,(a,b,c,d),wp(e);
  map f=h1,a,b,c,d;
  ideal i0=std(f(i));
```

See Section 4.6 [ideal], page 80; Section 4.9 [intvec], page 93; Section 5.1.124 [qhweight], page 246.

5.1.173 weightKB

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Syntax: `weightKB` (module_expression, int_expression , list_expression)
`weightKB` (ideal_expression, int_expression, list_expression)

Return: the same as the input type of the first argument

Purpose: If `I,d,wim` denotes the three arguments then weightKB computes the weighted degree-`d` part of a vector space basis (consisting of monomials) of the quotient ring, resp. of the quotient module, modulo `I` w.r.t. weights given by `wim` The information about the weights is given as a list of two intvec: `wim[1]` weights for all variables (positive), `wim[2]` weights for the module generators.

Note: This is a generalization of the command `kbase` with the same first two arguments.

Example:

```
  ring R=0, (x,y), wp(1,2);
weightKB(ideal(0),3,intvec(1,2));
↦ _[1]=x3
↦ _[2]=xy
```

See also: Section 5.1.69 [kbase], page 205.

5.1.174 write

Syntax: `write` (link_expression, expression_list)
for DBM links:
`write` (link, string_expression, string_expression)
`write` (link, string_expression)

Type: none

Purpose: writes data to a link.
If the link is of type `ASCII`, all expressions are converted to strings (and separated by a newline character) before they are written. As a consequence, only such values which can be converted to a string can be written to an `ASCII` link.
For ssi links, ring-dependent expressions are written together with a ring description. To prevent an evaluation of the expression before it is written, the `quote` command (possibly together with `eval`) can be used. A `write` blocks (i.e., does not return to the prompt), as long as a ssi link is not ready for writing.
For DBM links, `write` with three arguments inserts the first string as key and the second string as value into the dbm data base.
Called with two arguments, it deletes the entry with the key specified by the string from the data base.

Example:

```
// write the values of the variables f and i as strings into
// the file "outfile" (overwrite it, if it exists)
write(":w outfile",f,i);

// now append the string "that was f,i" (without the quotes)
// at the end of the file "outfile"
write(":a outfile","that was f,i");
// alternatively, links could be used:
link l=":a outfile"; l;
// type : ASCII
// mode : a
// name : outfile
// open : no
// read : not ready
// write: not ready
write(l," that was f,i");
// saving and retrieving data (ASCII format):
ring r=32003,(x,y,z),dp;
ideal i=x+y,z3+22y;
write(":w save_i",i);// this writes x+y,z3+22y to the file save_i
ring r=32003,(x,y,z),dp;
string s=read("save_i");   //creates the string x+y,z3+22y
execute("ideal k="+s+";"); // this defines an ideal k which
                           // is equal to i.
// for large objects, the ssi format and ssi links are better:
write("ssi:w save_i.ssi",i);
def j=read("ssi:r save_i.ssi");
```

See Chapter 4 [Data types], page 73; Section 5.1.27 [dump], page 175; Section 5.1.29 [eval], page 177; Section 4.10 [link], page 96; Section 5.1.120 [print], page 242; Section 5.1.121 [printf], page 244; Section 5.1.126 [quote], page 247; Section 5.1.130 [read], page 250; Section 5.3.7 [short], page 305.

5.2 Control structures

A sequence of commands surrounded by curly brackets (`{` and `}`) is a so-called block. Blocks are used in Singular in order to define procedures and to collect commands belonging to `if`, `else`, `for` and `while` statements and to the `example` part in libraries. Even if the sequence of statements consists of only a single command it has to be surrounded by curly brackets! Variables which are defined inside a block are not local to that block. Note that there need not be an ending semicolon at the end of the block.

Example:

```
if ( i>j )
{
  // This is the block
  int temp;
  temp=i;
  i=j;
  j=temp;
  kill temp;
}
```

5.2.1 apply

Syntax: `apply(` expression `,` function `);`

Purpose: applies the function to all elements of the first argument. The first argument must be of type `intvec`, `intmat`, or `list`. The result will be an expression list, its type and format will be set by the following assign. The function must be a kernel command or a procedure which takes one argument and returns a a value.

Example:

```
proc p(int x) {return(x^2);}
intvec v=1,2,3;
apply(v,p);
↦ 1 4 9
intvec vv=apply(v,p);vv;
↦ 1,4,9
```

```
list ll=apply(v,p);ll;
↦ [1]:
↦    1
↦ [2]:
↦    4
↦ [3]:
↦    9
```

5.2.2 break

Syntax: `break;`

Purpose: leaves the innermost `for` or `while` block.

Example:

```
while (1)
{
  ...
  if ( ... )
  {
    break; // leave the while block
  }
}
```

See Section 5.2 [Control structures], page 290; Section 5.2.8 [for], page 295; Section 5.2.15 [while], page 301.

5.2.3 breakpoint

Syntax: `breakpoint(` proc_name `);`
`breakpoint(` proc_name`,` line_no `);`

Purpose: sets a breakpoint at the beginning of the specified procedure or at the given line.
Note: Line number 1 is the first line of a library (for procedures from libraries), resp. the line with the `{`.
A line number of -1 removes all breakpoint from that procedure.

Example:

```
breakpoint(groebner);
↦ breakpoint 1, at line 831 in groebner
breakpoint(groebner, 176);
↦ breakpoint 2, at line 176 in groebner
breakpoint(groebner, -1);
↦ breakpoints in groebner deleted(0x6)
```

See Section 3.9.3 [Source code debugger], page 69; Section 5.2.16 [~], page 302.

5.2.4 continue

Syntax: `continue;`

Purpose: skips the rest of the innermost `for` or `while` loop und jumps to the beginning of the block. This command is only valid inside a `for` or a `while` construction.

Note: Unlike the C-construct it **does not execute the increment statement**. The command `continue` is mainly for internal use.

Example:

```
for (int i = 1 ; i<=10; i=i+1)
{
   ...
   if (i==3) { i=8;continue; }
     // skip the rest if i is 3 and
     // continue with the next i: 8
   i;
}
↦ 1
↦ 2
↦ 8
↦ 9
↦ 10
```

See Section 5.2 [Control structures], page 290; Section 5.2.8 [for], page 295; Section 5.2.15 [while], page 301.

5.2.5 else

Syntax: `if` (boolean_expression) true_block `else` false_block

Purpose: executes false_block if the boolean_expression of the `if` statement is false. This command is only valid in combination with an `if` command.

Example:

```
int i=3;
if (i > 5)
{
  "i is bigger than 5";
}
else
{
  "i is smaller than 6";
}
↦ i is smaller than 6
```

See Section 5.2 [Control structures], page 290; Section 4.7.5 [boolean expressions], page 89; Section 5.2.9 [if], page 296.

5.2.6 export

Syntax: `export` name ;
`export` list_of_names ;

Purpose: converts a local variable of a procedure to a global one, that is the identifier is not removed at the end of the procedure. However, the package the variable belongs to is not changed.

Note: Objects defined in a ring are not automatically exported when exporting the ring.

Example:

```
proc p1
{
  int i,j;
  export(i);
  intmat m;
  listvar();
  export(m);
}
p1();
↦ // m                         [1]  intmat 1 x 1
↦ // j                         [1]  int 0
↦ // i                         [0]  int 0
listvar();
↦ // m                         [0]  intmat 1 x 1
↦ // i                         [0]  int 0
```

See Section 5.2.7 [exportto], page 293; Section 5.2.10 [importfrom], page 296; Section 5.2.11 [keepring], page 298.

5.2.7 exportto

Syntax: `exportto(` package_name , name `);`
`exportto(` package_name , list_of_names `);`

Purpose: transfers an identifier in the current package into the one specified by package_name. package_name can be `Current`, `Top` or any other identifier of type package.

Note: Objects defined in a ring are not automatically exported when exporting the ring.

Warning: The identifier is transferred to the other package. It does no longer exist in the current package. If the identifier should only be copied, Section 5.2.10 [importfrom], page 296 should be used instead.

Example:

```
proc p1
{
  int i,j;
  exportto(Current,i);
  intmat m;
  listvar(Current);
  exportto(Top,m);
}
p1();
↦ // Top                                 [0]  package Top (T)
↦ // ::m                                 [1]  intmat 1 x 1
↦ // ::i                                 [0]  int 0
↦ // ::j                                 [1]  int 0
↦ // ::#                                 [1]  list, size: 0
↦ // ::p1                                [0]  proc
↦ // ::mathicgb_prOrder                  [0]  proc from singmathic.so (C)
↦ // ::mathicgb                          [0]  proc from singmathic.so (C)
↦ // ::create_ring                       [0]  proc from standard.lib
↦ // ::min                               [0]  proc from standard.lib
↦ // ::max                               [0]  proc from standard.lib
↦ // ::datetime                          [0]  proc from standard.lib
↦ // ::weightKB                          [0]  proc from standard.lib
↦ // ::fprintf                           [0]  proc from standard.lib
↦ // ::printf                            [0]  proc from standard.lib
↦ // ::sprintf                      [0]  proc from standard.lib
↦ // ::quotient4                    [0]  proc from standard.lib
↦ // ::quotient5                    [0]  proc from standard.lib
↦ // ::quotient3                    [0]  proc from standard.lib
↦ // ::quotient2                    [0]  proc from standard.lib
↦ // ::quotient1                    [0]  proc from standard.lib
↦ // ::quot                         [0]  proc from standard.lib
↦ // ::res                          [0]  proc from standard.lib
↦ // ::groebner                     [0]  proc from standard.lib
↦ // ::qslimgb                      [0]  proc from standard.lib
↦ // ::hilbRing                     [0]  proc from standard.lib
↦ // ::par2varRing                  [0]  proc from standard.lib
↦ // ::quotientList                 [0]  proc from standard.lib
↦ // ::stdhilb                      [0]  proc from standard.lib
↦ // ::stdfglm                      [0]  proc from standard.lib
↦ // ::Float                        [0]  proc from kernel (C)
↦ // ::crossprod                    [0]  proc from kernel (C)
package Test1;
exportto(Test1,p1);
listvar(Top);
↦ // Top                            [0]  package Top (T)
↦ // ::m                            [0]  intmat 1 x 1
↦ // ::i                            [0]  int 0
↦ // ::mathicgb_prOrder             [0]  proc from singmathic.so (C)
↦ // ::mathicgb                     [0]  proc from singmathic.so (C)
↦ // ::create_ring                  [0]  proc from standard.lib
↦ // ::min                          [0]  proc from standard.lib
↦ // ::max                          [0]  proc from standard.lib
↦ // ::datetime                     [0]  proc from standard.lib
↦ // ::weightKB                     [0]  proc from standard.lib
↦ // ::fprintf                      [0]  proc from standard.lib
↦ // ::printf                       [0]  proc from standard.lib
↦ // ::sprintf                      [0]  proc from standard.lib
↦ // ::quotient4                    [0]  proc from standard.lib
↦ // ::quotient5                    [0]  proc from standard.lib
↦ // ::quotient3                    [0]  proc from standard.lib
↦ // ::quotient2                    [0]  proc from standard.lib
↦ // ::quotient1                    [0]  proc from standard.lib
↦ // ::quot                         [0]  proc from standard.lib
↦ // ::res                          [0]  proc from standard.lib
↦ // ::groebner                     [0]  proc from standard.lib
↦ // ::qslimgb                      [0]  proc from standard.lib
↦ // ::hilbRing                     [0]  proc from standard.lib
↦ // ::par2varRing                  [0]  proc from standard.lib
↦ // ::quotientList                 [0]  proc from standard.lib
↦ // ::stdhilb                      [0]  proc from standard.lib
↦ // ::stdfglm                      [0]  proc from standard.lib
↦ // ::Float                        [0]  proc from kernel (C)
↦ // ::crossprod                    [0]  proc from kernel (C)
listvar(Test1);
↦ // Test1                          [0]  package Test1 (N)
↦ // ::p1                           [0]  proc
Test1::p1();
```

```
↦ // Test1                         [0]  package Test1 (N)
↦ // ::m                           [1]  intmat 1 x 1
↦ // ::i                           [0]  int 0
↦ // ::j                           [1]  int 0
↦ // ::#                           [1]  list, size: 0
↦ // ::p1                          [0]  proc
↦ // ** redefining m (  exportto(Top,m);)
listvar(Top);
↦ // Top                           [0]  package Top (T)
↦ // ::m                           [0]  intmat 1 x 1
↦ // ::i                           [0]  int 0
↦ // ::mathicgb_prOrder            [0]  proc from singmathic.so (C)
↦ // ::mathicgb                    [0]  proc from singmathic.so (C)
↦ // ::create_ring                 [0]  proc from standard.lib
↦ // ::min                         [0]  proc from standard.lib
↦ // ::max                         [0]  proc from standard.lib
↦ // ::datetime                    [0]  proc from standard.lib
↦ // ::weightKB                    [0]  proc from standard.lib
↦ // ::fprintf                     [0]  proc from standard.lib
↦ // ::printf                      [0]  proc from standard.lib
↦ // ::sprintf                     [0]  proc from standard.lib
↦ // ::quotient4                   [0]  proc from standard.lib
↦ // ::quotient5                   [0]  proc from standard.lib
↦ // ::quotient3                   [0]  proc from standard.lib
↦ // ::quotient2                   [0]  proc from standard.lib
↦ // ::quotient1                   [0]  proc from standard.lib
↦ // ::quot                        [0]  proc from standard.lib
↦ // ::res                         [0]  proc from standard.lib
↦ // ::groebner                    [0]  proc from standard.lib
↦ // ::qslimgb                     [0]  proc from standard.lib
↦ // ::hilbRing                    [0]  proc from standard.lib
↦ // ::par2varRing                 [0]  proc from standard.lib
↦ // ::quotientList                [0]  proc from standard.lib
↦ // ::stdhilb                     [0]  proc from standard.lib
↦ // ::stdfglm                     [0]  proc from standard.lib
↦ // ::Float                       [0]  proc from kernel (C)
↦ // ::crossprod                   [0]  proc from kernel (C)
listvar(Test1);
↦ // Test1                         [0]  package Test1 (N)
↦ // ::i                           [0]  int 0
↦ // ::p1                          [0]  proc
```

See Section 5.2.6 [export], page 292; Section 5.2.10 [importfrom], page 296; Section 5.2.11 [keepring], page 298.

5.2.8 for

Syntax: `for (` init_command`;` boolean_expression`;` iterate_commands`)` block

Purpose: repetitive, conditional execution of a command block.
The command init_command is executed first. Then boolean_expression is evaluated. If its value is TRUE the block is executed, otherwise the `for` statement is complete. After each execution of the block, the command iterate_command is executed and boolean_expression is evaluated. This is repeated until boolean_expression evaluates to

FALSE.
The command `break;` leaves the innermost `for` construct.

Example:

```
// sum of 1 to 10:
int s=0;
for (int i=1; i<=10; i=i+1)
{
   s=s+i;
}
s;
↦ 55
```

See Section 5.2 [Control structures], page 290; Section 4.7.5 [boolean expressions], page 89; Section 5.2.2 [break], page 291; Section 5.2.4 [continue], page 291; Section 5.2.9 [if], page 296; Section 5.2.15 [while], page 301.

5.2.9 if

Syntax: `if (` boolean_expression `)` true_block
`if (` boolean_expression `)` true_block `else` false_block

Purpose: executes true_block if the boolean condition is true. If the `if` statement is followed by an `else` statement and the boolean condition is false, then false_block is executed.

Example:

```
int i = 9;
matrix m[i][i];
if (i > 5 and typeof(m) == "matrix")
{
  m[i][i] = i;
}
```

See Section 5.2 [Control structures], page 290; Section 4.7.5 [boolean expressions], page 89; Section 5.2.2 [break], page 291; Section 5.2.5 [else], page 292.

5.2.10 importfrom

Syntax: `importfrom(` package_name `,` name `);`
`importfrom(` package_name `,` list_of_names `);`

Purpose: creates a new identifier in the current package which is a copy of the one specified by name in the package package_name. package_name can be `Top` or any other identifier of type package.

Note: Objects defined in a ring are not automatically imported when importing the ring.

Warning: The identifier is copied to the current package. It does still exist (independently) in the package package_name. If the identifier should be erased in the package from which it originates, Section 5.2.7 [exportto], page 293 should be used instead.

Example:

```
listvar(Top);
↦ // Top                          [0]  package Top (T)
↦ // ::mathicgb_prOrder           [0]  proc from singmathic.so (C)
↦ // ::mathicgb                   [0]  proc from singmathic.so (C)
↦ // ::create_ring                  [0]  proc from standard.lib
↦ // ::min                          [0]  proc from standard.lib
↦ // ::max                          [0]  proc from standard.lib
↦ // ::datetime                     [0]  proc from standard.lib
↦ // ::weightKB                     [0]  proc from standard.lib
↦ // ::fprintf                      [0]  proc from standard.lib
↦ // ::printf                       [0]  proc from standard.lib
↦ // ::sprintf                      [0]  proc from standard.lib
↦ // ::quotient4                    [0]  proc from standard.lib
↦ // ::quotient5                    [0]  proc from standard.lib
↦ // ::quotient3                    [0]  proc from standard.lib
↦ // ::quotient2                    [0]  proc from standard.lib
↦ // ::quotient1                    [0]  proc from standard.lib
↦ // ::quot                         [0]  proc from standard.lib
↦ // ::res                          [0]  proc from standard.lib
↦ // ::groebner                     [0]  proc from standard.lib
↦ // ::qslimgb                      [0]  proc from standard.lib
↦ // ::hilbRing                     [0]  proc from standard.lib
↦ // ::par2varRing                  [0]  proc from standard.lib
↦ // ::quotientList                 [0]  proc from standard.lib
↦ // ::stdhilb                      [0]  proc from standard.lib
↦ // ::stdfglm                      [0]  proc from standard.lib
↦ // ::Float                        [0]  proc from kernel (C)
↦ // ::crossprod                    [0]  proc from kernel (C)
load("inout.lib");
listvar(Top);
↦ // Top                            [0]  package Top (T)
↦ // ::mathicgb_prOrder             [0]  proc from singmathic.so (C)
↦ // ::mathicgb                     [0]  proc from singmathic.so (C)
↦ // ::create_ring                  [0]  proc from standard.lib
↦ // ::min                          [0]  proc from standard.lib
↦ // ::max                          [0]  proc from standard.lib
↦ // ::datetime                     [0]  proc from standard.lib
↦ // ::weightKB                     [0]  proc from standard.lib
↦ // ::fprintf                      [0]  proc from standard.lib
↦ // ::printf                       [0]  proc from standard.lib
↦ // ::sprintf                      [0]  proc from standard.lib
↦ // ::quotient4                    [0]  proc from standard.lib
↦ // ::quotient5                    [0]  proc from standard.lib
↦ // ::quotient3                    [0]  proc from standard.lib
↦ // ::quotient2                    [0]  proc from standard.lib
↦ // ::quotient1                    [0]  proc from standard.lib
↦ // ::quot                         [0]  proc from standard.lib
↦ // ::res                          [0]  proc from standard.lib
↦ // ::groebner                     [0]  proc from standard.lib
↦ // ::qslimgb                      [0]  proc from standard.lib
↦ // ::hilbRing                     [0]  proc from standard.lib
↦ // ::par2varRing                  [0]  proc from standard.lib
↦ // ::quotientList                 [0]  proc from standard.lib
↦ // ::stdhilb                      [0]  proc from standard.lib
↦ // ::stdfglm                      [0]  proc from standard.lib
↦ // ::Float                        [0]  proc from kernel (C)
↦ // ::crossprod                    [0]  proc from kernel (C)
```

```
importfrom(Inout,pause);
listvar(Top);
↦ // Top                             [0]  package Top (T)
↦ // ::pause                         [0]  proc from inout.lib
↦ // ::mathicgb_prOrder              [0]  proc from singmathic.so (C)
↦ // ::mathicgb                      [0]  proc from singmathic.so (C)
↦ // ::create_ring                   [0]  proc from standard.lib
↦ // ::min                           [0]  proc from standard.lib
↦ // ::max                           [0]  proc from standard.lib
↦ // ::datetime                      [0]  proc from standard.lib
↦ // ::weightKB                      [0]  proc from standard.lib
↦ // ::fprintf                       [0]  proc from standard.lib
↦ // ::printf                        [0]  proc from standard.lib
↦ // ::sprintf                       [0]  proc from standard.lib
↦ // ::quotient4                     [0]  proc from standard.lib
↦ // ::quotient5                     [0]  proc from standard.lib
↦ // ::quotient3                     [0]  proc from standard.lib
↦ // ::quotient2                     [0]  proc from standard.lib
↦ // ::quotient1                     [0]  proc from standard.lib
↦ // ::quot                          [0]  proc from standard.lib
↦ // ::res                           [0]  proc from standard.lib
↦ // ::groebner                      [0]  proc from standard.lib
↦ // ::qslimgb                       [0]  proc from standard.lib
↦ // ::hilbRing                      [0]  proc from standard.lib
↦ // ::par2varRing                   [0]  proc from standard.lib
↦ // ::quotientList                  [0]  proc from standard.lib
↦ // ::stdhilb                       [0]  proc from standard.lib
↦ // ::stdfglm                       [0]  proc from standard.lib
↦ // ::Float                         [0]  proc from kernel (C)
↦ // ::crossprod                     [0]  proc from kernel (C)
```

See Section 5.2.6 [export], page 292; Section 5.2.7 [exportto], page 293; Section 5.2.11 [keepring], page 298.

5.2.11 keepring

Syntax: `keepring` name ;

Warning: This command is obsolete. Instead the respective identifiers in the ring should be exported and the ring itself should subsequently be returned. The command is only included for backward compatibility and may be removed in future releases.

Purpose: moves the specified ring to the next (upper) level. This command can only be used inside of procedures and it should be the last command before the `return` statement. There it provides the possibility to keep a ring which is local to the procedure (and its objects) accessible after the procedure ended without making the ring global.

Example:

```
proc P1
{
  ring r=0,x,dp;
  keepring r;
}
proc P2
```

```
{
  "inside P2: " + nameof(basering);
  P1();
  "inside P2, after call of P1: " + nameof(basering);
}
ring r1= 0,y,dp;
P2();
↦ inside P2: r1
↦ inside P2, after call of P1: r
"at top level: " + nameof(basering);
↦ at top level: r1
```

See Section 4.20 [ring], page 127.

5.2.12 load

Syntax: `load(` string_expression `);`
`load(` string_expression `,` string_expression `);`

Type: none

Purpose: reads a library of procedures from a file. In contrast to the command `LIB` (see note below), the command `load` does not add the procedures of the library to the package `Top`, but only to the package corresponding to the library. If the given filename does not start with `.` or `/`, the following directories are searched for it (in the given order): the current directory, the directories given in the environment variable `SINGULARPATH`, some default directories relative to the location of the Singular executable program, and finally some default absolute directories. You can view the search path which Singular uses to locate its libraries, by starting up Singular with the option `-v`, or by issuing the command `system("with");`.

The second string selections options for loading.

Note: `load(<string_expr>,"with") is equivalent to`
`LIB <string_expr>`.

Note: `load(<string_expr>,"try") is equivalent to`
`LIB <string_expr>` which never fails - test the package name to distinguish.

All loaded libraries are displayed by the `listvar(package);` command:

```
  option(loadLib);   // show loading of libraries;
                     // standard.lib is loaded
  listvar(package);
↦ // Singmathic                    [0]  package Singmathic (C,singmathic.s\
  o)
↦ // Standard                      [0]  package Standard (S,standard.lib)
↦ // Top                           [0]  package Top (T)
                     // the names of the procedures of inout.lib
  load("inout.lib"); // are now known to Singular
↦ // ** loaded inout.lib (4.1.2.0,Feb_2019)
  listvar(package);
↦ // Inout                         [0]  package Inout (S,inout.lib)
↦ // Singmathic                    [0]  package Singmathic (C,singmathic.s\
  o)
↦ // Standard                      [0]  package Standard (S,standard.lib)
↦ // Top                           [0]  package Top (T)
```

```
     load("blabla.lib","try");
     listvar(package);
   ↦ // Inout                          [0]  package Inout (S,inout.lib)
   ↦ // Singmathic                     [0]  package Singmathic (C,singmathic.s\
     o)
   ↦ // Standard                       [0]  package Standard (S,standard.lib)
   ↦ // Top                            [0]  package Top (T)
     option(noloadLib);   // do not show loading of libraries;
     load("matrix.lib","try");
     listvar(package);
   ↦ // Elim                           [0]  package Elim (S,elim.lib)
   ↦ // Triang                         [0]  package Triang (S,triang.lib)
   ↦ // Absfact                        [0]  package Absfact (S,absfact.lib)
   ↦ // Primdec                        [0]  package Primdec (S,primdec.lib)
   ↦ // Ring                           [0]  package Ring (S,ring.lib)
   ↦ // Random                         [0]  package Random (S,random.lib)
   ↦ // General                        [0]  package General (S,general.lib)
   ↦ // Polylib                        [0]  package Polylib (S,polylib.lib)
   ↦ // Nctools                        [0]  package Nctools (S,nctools.lib)
   ↦ // Matrix                         [0]  package Matrix (S,matrix.lib)
   ↦ // Inout                          [0]  package Inout (S,inout.lib)
   ↦ // Singmathic                     [0]  package Singmathic (C,singmathic.s\
     o)
   ↦ // Standard                       [0]  package Standard (S,standard.lib)
   ↦ // Top                            [0]  package Top (T)
```

Each time a library (Section 3.8 [Libraries], page 55) / dynamic module (Section 3.10 [Dynamic loading], page 71) is loaded, the corresponding package is created, if it does not already exist.

The name of a package corresponding to a Singular library is derived from the name of the library file. The first letter is capitalized and everything to right of the left-most dot is dropped. For a dynamic module the packagename is hard-coded in the binary file.

Only the names of the procedures in the library are loaded, the body of the procedures is read during the first call of this procedure. This minimizes memory consumption by unused procedures. When Singular is started with the -q or --quiet option, no message about the loading of a library is displayed.

```
     option(loadLib); // show loading of libraries; standard.lib is loaded
                      // the names of the procedures of inout.lib
     load("inout.lib"); // are now known to Singular
   ↦ // ** loaded inout.lib (4.1.2.0,Feb_2019)
     listvar();
```

See Section 3.1.6 [Command line options], page 19; Section A.1.9 [Dynamic modules], page 708; Section 5.1.79 [LIB], page 211; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 793; Section 5.2.7 [exportto], page 293; Section 5.2.10 [importfrom], page 296; Section 4.16 [package], page 119; Section 4.18 [proc], page 124; Section D.1 [standard_lib], page 793; Section 4.22 [string], page 130; Section 5.1.155 [system], page 275.

5.2.13 quit

Syntax: `exit;`
`quit;`

Purpose: quits Singular; works also from inside a procedure or from an interrupt. Instead of `quit`, the synonymous command `exit` may be used.

Example:

```
quit;
```

5.2.14 return

Syntax: `return` (expression_list) `;`
`return ();`

Type: any

Purpose: returns the result(s) of a procedure and can only be used inside a procedure. Note that the brackets are required even if no return value is given.

Example:

```
proc p2
{
  int i,j;
  for(i=1;i<=10;i++)
  {
    j=j+i;
  }
  return(j);
}
// can also return an expression list, i.e., more than one value
proc tworeturn ()
{ return (1,2); }
int i,j = tworeturn();
// return type may even depend on the input
proc type_return (int i)
{
  if (i > 0) {return (i);}
  else {return (list(i));}
}
// then we need def type (or list) to collect value
def t1 = type_return(1);
def t2 = type_return(-1);
```

See Chapter 4 [Data types], page 73; Section 4.18 [proc], page 124.

5.2.15 while

Syntax: `while` (boolean_expression) block

Purpose: repetitive, conditional execution of block.
The boolean_expression is evaluated and if its value is TRUE, the block gets executed. This is repeated until boolean_expression evaluates to FALSE. The command `break` leaves the innermost `while` construction.

Example:

```
int i = 9;
while (i>0)
{
   // ... // do something for i=9, 8, ..., 1
   i = i - 1;
}
```

```
while (1)
{
   // ...   // do something forever
   if (i == -5) // but leave the loop if i is -5
   {
     break;
   }
}
```

See Section 5.2 [Control structures], page 290; Section 4.7.5 [boolean expressions], page 89; Section 5.2.2 [break], page 291.

5.2.16 ~ (break point)

Syntax: `~;`

Purpose: sets a break point. Whenever SINGULAR reaches the command `~;` in a sequence of commands it prompts for input. The user may now input lines of SINGULAR commands. The line length cannot exceed 80 characters. SINGULAR proceeds with the execution of the command following `~;` as soon as it receives an empty line.
Furthermore, the debug mode will be activated: See Section 3.9.3 [Source code debugger], page 69.

Example:

```
proc t
{
  int i=2;
  ~;
  return(i+1);
}
t();
↦ -- break point in t --
↦ -- 0: called    from STDIN --
// here local variables of the procedure can be accessed
i;
↦ 2
↦ -- break point in t --

↦ 3
```

See Section 3.9.4 [Break points], page 70.

5.3 System variables

5.3.1 degBound

Type: int

Purpose: The standard basis computation is stopped if the total (weighted) degree exceeds `degBound` - used in `std`, `slimgb`, `system("verifyGB",..)`
`degBound` should not be used for a global ordering with inhomogeneous input, if the ordering is not `dp` or `Dp`. (Remark: elimination requires always an eliminiation ordering).

Reset this bound by setting degBound to 0.
The exact meaning of "degree" depends on the ring ordering and the command: slimgb uses always the total degree with weights 1, std does so for block orderings, only.

Example:

```
degBound = 7;
option();
↦ //options for 'std'-command: degBound
ideal j=std(i);
degBound;
↦ 7
degBound = 0; //resets degree bound to infinity
```

See Section 5.1.19 [deg], page 170; Section 4.7 [int], page 85; Section 5.1.111 [option], page 234; Section 5.1.151 [std], page 271; Section 5.1.155 [system], page 275.

5.3.2 echo

Type: int

Purpose: input is echoed if echo >= voice.
echo is a local setting for a procedure and defaulted to 0.
echo does not affect the output of commands.

Example:

```
echo = 1;
int i = echo;
↦ int i = echo;
```

See Section 4.7 [int], page 85; Section 5.3.11 [voice], page 308.

5.3.3 minpoly

Type: number

Purpose: describes the coefficient field of the current basering as an algebraic extension with the minimal polynomial equal to minpoly. Setting the minpoly should be the first command after defining the ring.

Note: The minimal polynomial has to be specified in the syntax of a polynomial. Its variable is not one of the ring variables, but the algebraic element which is being adjoined to the field. Algebraic extensions in Singular are only possible over the rational numbers or over Z/p, p a prime number.
Singular does not check whether the given polynomial is irreducible! It can be checked in advance with the function factorize (see Section 5.1.36 [factorize], page 180).

Example:

```
  //(Q[i]/(i^2+1))[x,y,z]:
  ring Cxyz=(0,i),(x,y,z),dp;
  minpoly=i^2+1;
  i2;  //this is a number, not a poly
↦ -1
```

See Section 5.1.36 [factorize], page 180; Section 4.20 [ring], page 127.

5.3.4 multBound

Type: int

Purpose: The standard basis computation is stopped if the ideal is zero-dimensional in a ring with local ordering and its multiplicity (mult) is lower than multBound.
Reset this bound by setting multBound to 0.

Example:

```
ring r=0,(x,y,z),ds;
ideal i,j;
i=x7+y7+z6,x6+y8+z7,x7+y5+z8,
x2y3+y2z3+x3z2,x3y2+y3z2+x2z3;
multBound=100;
j=std(i);
degree(j);
↦ // dimension (local)   = 0
↦ // multiplicity = 98
multBound=0;  //disables multBound
j=std(i);
degree(j);
↦ // dimension (local)   = 0
↦ // multiplicity = 86
```

See Section 4.7 [int], page 85; Section 5.1.101 [mult], page 228; Section 5.1.111 [option], page 234; Section 5.1.151 [std], page 271.

5.3.5 noether

Type: poly

Purpose: The standard basis computation in local rings cuts off all monomials above (in the sense of the monomial ordering) the monomial noether during the computation.
Reset noether by setting noether to 0.

Example:

```
  ring R=32003,(x,y,z),ds;
  ideal i=x2+y12,y13;
  std(i);
↦ _[1]=x2+y12
↦ _[2]=y13
  noether=x11;
  std(i);
↦ _[1]=x2
  noether=0; //disables noether
```

See Section 4.17 [poly], page 120; Section 5.1.151 [std], page 271.

5.3.6 printlevel

Type: int

Purpose: sets the debug level for dbprint. If printlevel >= voice then dbprint is equivalent to print, otherwise nothing is printed.

Note: See Section 3.8.6 [Procedures in a library], page 57, for a small example about how this is used for the display of comments while procedures are executed.

Example:

```
  voice;
↦ 1
  printlevel=0;
  dbprint(1);
  printlevel=voice;
  dbprint(1);
↦ 1
```

See Section 5.1.17 [dbprint], page 169; Section 4.7 [int], page 85; Section 5.3.11 [voice], page 308.

5.3.7 short

Type: int

Purpose: the output of monomials is done in the short manner, if `short` is non-zero. A C-like notion is used, if short is zero. Both notations may be used as input.
The default depends on the names of the ring variables (0 if there are names of variables longer than 1 character, 1 otherwise). Every change of the basering sets `short` to the previous value for that ring. In other words, the value of the variable `short` is "ring-local".
If the names are long, or the ring non-commutative, `short` can not be changed to 1.

Example:

```
  ring r=23,x,dp;
  int save=short;
  short=1;
  2x2,x2;
↦ 2x2 x2
  short=0;
  2x2,x2;
↦ 2*x^2 x^2
  short=save;  //resets short to the previous value
```

See Section 4.7 [int], page 85.

5.3.8 timer

Type: int

Purpose:

1. the CPU time (i.e, user and system time) used for each command is printed if timer >0 , if this time is bigger than a (customizable) minimal time and if `printlevel+1 >= voice` (which is by default true on the SINGULAR top level, but not true while procedures are executed).
2. yields the CPU time used since the start-up of SINGULAR in a (customizable) resolution.

The default setting of `timer` is 0, the default minimal time is 0.5 seconds, and the default timer resolution is 1 (i.e., the default unit of time is one second). The minimal time and timer resolution can be set using the command line options `--min-time` and

`--ticks-per-sec` and can be checked using `system("--min-time")` and `system("--ticks-per-sec")`.

How to use `timer` in order to measure the time for a sequence of commands, see example below.

Note for Windows95/98:

The value of the `timer` cannot be used (resp. trusted) when Singular is run under Windows95/98 (this is due to the shortcomings of the Windows95/98 operating system). Use Section 5.3.10 [rtimer], page 308, instead.

Example:

```
  timer=1; // The time of each command is printed
  int t=timer; // initialize t by timer
  ring r=0,(x,y,z),dp;
  poly p=(x+2y+3z+4xy+5xz+6yz)^20;
  // timer as int_expression:
  t=timer-t;
  t;  // yields the time in ticks-per-sec (default 1)
↦ 0
      // since t was initialized by timer
  int tps=system("--ticks-per-sec");
  t div tps; // yields the time in seconds truncated to int
↦ 0
  timer=0;
  system("--ticks-per-sec",1000); // set timer resolution to ms
  t=timer; // initialize t by timer
  p=(x+2y+3z+4xy+5xz+6yz)^20;
  timer-t;  // time in ms
↦ 30
```

See Section 3.1.6 [Command line options], page 19; Section 5.3.6 [printlevel], page 304; Section 5.3.10 [rtimer], page 308; Section 5.1.155 [system], page 275; Section 5.3.11 [voice], page 308.

5.3.9 TRACE

Type: int

Purpose: sets level of debugging.

`TRACE=0` No debugging messages are printed.

`TRACE=1` Messages about entering and leaving of procedures are displayed.

`TRACE=3` Messages about entering and leaving of procedures together with line numbers are displayed.

`TRACE=4` Each line is echoed and the interpretation of commands in this line is suspended until the user presses `RETURN`.

`TRACE=8` (debug version only:) show basering for all levels

`TRACE=128`
show all calls to kernel routines

`TRACE=256`
show all assigns

`TRACE=512`
: show all automatic type conversions

`TRACE=1024`
: profiling: print line numbers to smon.out

`TRACE` is defaulted to 0.
`TRACE` does not affect the output of commands.

Example:

```
  TRACE=1;
  LIB "general.lib";
  sum(1..100);
↦ entering sum (level 0)
↦ entering   lsum (level 1)
↦ entering     lsum (level 2)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ leaving      lsum (level 2)
↦ entering     lsum (level 2)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ leaving      lsum (level 2)
↦ leaving    lsum (level 1)
↦ entering   lsum (level 1)
↦ entering     lsum (level 2)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ leaving      lsum (level 2)
↦ entering     lsum (level 2)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ entering       lsum (level 3)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ entering         lsum (level 4)
↦ leaving          lsum (level 4)
↦ leaving        lsum (level 3)
↦ leaving      lsum (level 2)
↦ leaving    lsum (level 1)
↦ leaving  sum (level 0)
↦ 5050
```

See Section 4.7 [int], page 85.

5.3.10 rtimer

Type: int

Purpose: identical to `timer` (see Section 5.3.8 [timer], page 305), except that real times (i.e., wall-clock) times are reported, instead of CPU times. This can be trusted on all operating systems (including Windows95/98).

5.3.11 voice

Type: int

Purpose: shows the nesting level of procedures.

Note: See Section 3.8 [Libraries], page 55, for a small example how this is used for the display of comments while procedures are executed.

Example:

```
  voice;
↦ 1
proc p
{
  voice;
};
p();
↦ 2
```

See Section 5.1.17 [dbprint], page 169; Section 5.1.82 [listvar], page 213; Section 5.3.6 [printlevel], page 304.

6 Tricks and pitfalls

6.1 Limitations

Singular has the following limitations:

- the characteristic of a prime field must be less than or equal to 2147483647 (2^31)
(the characteristic of a prime field in the factory routines must be less than 536870912 (2^29))
(the characteristic of a prime field in the NTL routines must be less than NTL_SP_BOUND (2^30) on 32bit machines - This is always the case since currently, only factory uses NTL.)
- if only NTL is used, extgcd has a limit on the coefficients: NTL_MAX_FFTPRIMES // At the current setting of 20000, on 64-bit machines with 50-bit // FFT primes, this allows for polynomials with 20*50/2 = 500K-bit // coefficients, while the table itself takes 160KB.
- the number of elements in GF(p,n) must be less than 65536
- the (weighted) degree of a monomial must be less or equal than 2147483647
- the rank of any free module must be less or equal than 2147483647
- the maximal allowed exponent of a ring variable depends on the ordering of the ring and is at least 32767. See also Section B.2 [Monomial orderings], page 768 for setting other limits.
- the precision of long floating point numbers (for ground field `real`) must be less or equal than 32767
- integers (of type `int`) have the limited range from -2147483648 to 2147483647
- floating point numbers (type `number` from field `real`) have a limited range which is machine dependent. A typical range is -1.0e-38 to 1.0e+38. The string representation of overflow and underflow is machine dependent, as well. For example `"Inf"` on Linux, or `"+.+00e+00"` on HPUX.
Their input syntax is given by `scanf`, but must start with a digit.
- floating point numbers (type `number` from field `real` with a precision p larger then 3) use internally `mpf_set_default_prec(3.5*p+1)`.
Their input syntax is given by `mpf_set_str` from GMP, but must start with a digit.
- the length of an identifier is unlimited but `listvar` displays only the first 20 characters
- statements may not contain more than 10000 tokens
- tokens (i.e. strings, numbers, ...) may not be longer than 16382 characters
- All input to Singular must be 7-bit clean, i.e. special characters like the the German Umlaute (ä, ö, etc.), or the French accent characters may neither appear as input to Singular, nor in libraries or procedure definitions.
- `vspace`, used in `system("verifyGB",I)` and `farey` can not use more than MAX_PROCESS (64) cpus. This limit can be changed in the source file `kernel/oswrapper/vspace.h`.

6.2 System dependent limitations

Ports of Singular to different systems do not always implement all possible parts of Singular:

- dynamic modules are implemented for
  - unix systems with ELF format for executables (Linux, Solaris, FreeBSD)

6.3 Major differences to the C programming language

Although many constructs from Singular's programming language are similar to those from the C programming language, there are some subtle differences. Most notably:

6.3.1 No rvalue of increments and assignments

The increment operator ++ (resp. decrement operator --) has no rvalue, i.e., cannot be used on the right-hand sides of assignments. So, instead of

```
j = i++;  // WRONG!!!
```

(which results in an error), it must be written

```
i++; j = i;
```

Likewise, an assignment expression does not have a result. Therefore, compound assignments like `i = j = k;` are not allowed and result in an error.

6.3.2 Evaluation of logical expressions

All arguments of a logical expression are first evaluated and then the value of the logical expression is determined. For example, the logical expressions `(a || b)` is evaluated by first evaluating `a` *and* `b`, even though the value of `b` has no influence on the value of `(a || b)`, if `a` evaluates to true.

Note, that this evaluation is different from the left-to-right, conditional evaluation of logical expressions (as found in most programming languages). For example, in these other languages, the value of `(1 || b)` is determined without ever evaluating `b`. This causes some problems with boolean tests on variables, which might not be defined at evaluation time. For example, the following results in an error, if the variable `i` is undefined:

```
if (defined(i) && i > 0) {} // WRONG!!!
```

This must be written instead as:

```
if (defined(i))
{
  if (i > 0) {}
}
```

However, there are several short work-arounds for this problem:

1. If a variable (say, `i`) is only to be used as a boolean flag, then define (value is TRUE) and undefine (value is FALSE) `i` instead of assigning a value. Using this scheme, it is sufficient to simply write

   ```
   if (defined(i))
   ```

   in order to check whether `i` is TRUE. Use the command `kill` to undefine a variable, i.e. to assign it a FALSE value (see Section 5.1.71 [kill], page 206).

2. If a variable can have more than two values, then define it, if necessary, before it is used for the first time. For example, if the following is used within a procedure

   ```
   if (! defined(DEBUG)) { int DEBUG = 1;}
   ...
   if (DEBUG == 3)  {...}
   if (DEBUG == 2)  {...}
   ...
   ```

   then a user of this procedure does not need to care about the existence of the `DEBUG` variable – this remains hidden from the user. However, if `DEBUG` exists globally, then its local default value is overwritten by its global one.

6.3.3 No case or switch statement

SINGULAR does not offer a `case` (or `switch`) statement. However, it can be imitated in the following way:

```
while (1)
{
   if (choice == choice_1) { ...; break;}
   ...
   if (choice == choice_n) { ...; break;}
   // default case
   ...; break;
}
```

6.3.4 Usage of commas

In SINGULAR, a comma separates list elements and the value of a comma expression is a list. Hence, commas cannot be used to combine several expressions into a single expression. For example, instead of writing

```
for (i=1, j=5; i<5 || j<10; i++, j++) {...} // WRONG!!!!!!
```

one has to write

```
for (i,j = 1,5; i<5 || j<10; i++, j++) {...}
```

6.3.5 Usage of brackets

In SINGULAR, curly brackets (`{ }`) **must always** be used to enclose the statement body following such constructs like `if`, `else`, `for`, or `while`, even if this block consists of only a single statement. Similarly, in the return statement of a procedure, parentheses (`( )`) **must always** be used to enclose the return value. Even if there is no value to return, parentheses have to be used after a return statement (i.e., `return();`). For example,

```
if (i == 1) return i;    // WRONG!!!!!
```

results in an error. Instead, it must be written as

```
if (i == 1) { return (i); }.
```

6.3.6 Behavior of continue

SINGULAR's `continue` construct is only valid inside the body of a `for` or `while` construct. It skips the rest of the loop-body and jumps to the beginning of the block. Unlike the C-construct SINGULAR's `continue` **does not execute the increment statement**. For example,

```
for (int i = 1 ; i<=10; i=i+1)
{
   ...
   if (i==3) { i=8;continue; }
     // skip the rest if i is 3 and
     // continue with the next i: 8
   i;
}
↦ 1
↦ 2
↦ 8
↦ 9
↦ 10
```

6.3.7 Return type of procedures

Although the Singular language is a strongly typed programming language, the type of the return value of a procedure does not need to be specified. As a consequence, the return type of a procedure may vary, i.e., may, for example, depend on the input. However, the return value of such a procedure may then only be assigned to a variable of type `def`.

```
proc type_return (int i)
{
  if (i > 0) {return (i);}
  else {return (list(i));}
}
def t1 = type_return(1);
def t2 = type_return(-1);
typeof(t1); typeof(t2);
↦ int
↦ list
```

Furthermore, it is mandatory to assign the return value of a procedure to a variable of type `def`, if a procedure changes the current ring using the `keepring` command (see Section 5.2.11 [keepring], page 298) and returns a ring-dependent value (like a polynomial or module).

```
proc def_return
{
  ring r=0,(x,y),dp;
  poly p = x;
  keepring r;
  return (x);
}
def p = def_return();
// poly p = def_return(); would be WRONG!!!
typeof(p);
↦ poly
```

On the other hand, more than one value can be returned by a single `return` statement. For example,

```
proc tworeturn () { return (1,2); }
int i,j = tworeturn();
```

6.3.8 First index is 1

Although the Singular language is C like, the indices of all objects which may have an index start at 1.

```
  ring r;
  ideal i=1,x,z;
  i[2];
↦ x
  intvec v=1,2,3;
  v[1];
↦ 1
  poly p=x+y+z;
  p[2];
↦ y
  vector h=[x+y,x,z];
  h[1];
```

```
↦ x+y
  h[1][1];
↦ x
```

6.4 Miscellaneous oddities

1. integer division

   If two numerical constants (i.e., two sequences of digits) are divided using the / operator, the surrounding whitespace determines which division to use: if there is no space between the constants and the / operator (e.g., "3/2"), both numerical constants are treated as of type `number` and the current ring division is used. If there is at least one space surrounding the / operator (e.g., "3 / 2"), both numerical constants are treated as of type `int` and an integer division is performed. To avoid confusion, use the `div` operator instead of / for integer division and an explicit type cast to `number` for ring division. Note, that this problem does only occur for divisions of numerical constants. It also applies for large numerical constants which are of type `bigint`.

   ```
     ring r=32002,x,dp;
   ↦ // ** 32002 is invalid as characteristic of the ground field. 32003 is us\
      ed.
     3/2;    // ring division
   ↦ -16000
     3 / 2;  // integer division
   ↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  3 / 2;  // int\
      eger division<<
   ↦ 1
     3 div 2;
   ↦ 1
     number(3) / number(2);
   ↦ -16000
     number a=3;
     number b=2;
     a/b;
   ↦ -16000
     int c=3;
     int d=2;
     c / d;
   ↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  c / d;<<
   ↦ 1
   ```

2. monomials and precedence

   The formation of a monomial has precedence over all operators (a monomial is here an optional coefficient followed by any sequence of ring variables (possibly followed by an exponent) which only conssist of letters, digits and (over the rationals) / without any whitespace):

   ```
     ring r=0,(x,y),dp;
     2xy^2 == (2*x*y)^2;
   ↦ 1
     2xy^2 == 2x*y^2;
   ↦ 0
     2x*y^2 == 2*x * (y^2);
   ↦ 1
   ```

   During that formation no operator is involved: in the non-commutative case, we have

```
  LIB "nctools.lib";
  ring r = 0,(x,y),dp;
  def S = superCommutative();
  xy == yx;
↦ 1
  x*y == y*x;
↦ 1
  x*y, y*x;
↦ xy xy
```

3. meaning of `mult`

   For an arbitrary ideal or module `i`, `mult(i)` returns the multiplicity of the ideal generated by the leading monomials of the given generators of `i`, hence depends on the monomial ordering!

   A standard mistake is to interpret `degree(i)` or `mult(i)` for an inhomogeneous ideal `i` as the degree of the homogenization or as something like the 'degree of the affine part'. For the ordering `dp` (degree reverse lexicographical) the converse is true: if `i` is given by a standard basis, `mult(i)` is the degree of the homogeneous ideal obtained by homogenization of `i` and then putting the homogenizing variable to 0, hence it is the degree of the part at infinity (this can also be checked by looking at the initial ideal).

4. size of ideals

   `size` counts the non-zero entries of an ideal or module. Use `ncols` to determine the actual number of entries in the ideal or module.

5. computations in `qring`

   In order to speed up computations in quotient rings, SINGULAR usually does not reduce polynomials w.r.t. the quotient ideal; rather the given representative is used as long as possible during computations. If it is necessary, reduction is done during standard base computations. To reduce a polynomial `f` by hand w.r.t. the current quotient ideal use the command `reduce(f,std(0))` (see Section 5.1.131 [reduce], page 251).

6. degree of a polynomial

   `degBound`
   The exact meaning of "degree" depends on the ring odering and the command: `slimgb` uses always the total degree with weights 1, `std` does so only for block orderings.

   `hilb`
   the degree is the total degree with weights 1 unless a weight vector is given

   `kbase`
   the degree is the total degree with weights 1 (to use another weight vector see Section 5.1.173 [weightKB], page 288)

7. substring selection

   To extract substrings from a `string`, square brackets are used, enclosing either two comma-separated `int`s or an `intvec`. Although two comma-separated `int`s represent an `intvec`, they mean different things in substring access. Square brackets enclosing two `int`s (e.g. `s[2,6]`) return a substring where the first integer denotes the starting position and the second integer denotes the length of the substring. The result is returned as a `string`. Square brackets enclosing an `intvec` (e.g. `s[intvec(2,6)]`) return the characters of the string at the position given by the values of the `intvec`. The result is returned as an expression list of strings.

```
  string s = "one-word";
  s[2,6];     // a substring starting at the second char
↦ ne-wor
  size(_);
```

```
↦ 6
  intvec v = 2,6;
  s[v];      // the second and the sixth char
↦ n o
  string st = s[v];  // stick together by an assignment
  st;
↦ no
  size(_);
↦ 2
  v = 2,6,8;
  s[v];
↦ n o d
```

8. packages and indexed variables

   See example

```
package K;
string varok; exportto(K,varok);
string work(1); exportto(K,work(1));
int i(1..3); exportto(K,i(1..3));
// Toplevel does not contain i(1..3)
listvar();
// i(1..3) are stored in Package 'K'
listvar(K);
↦ // K                             [0]  package K (N)
↦ // ::i(3)                        [0]  int 0
↦ // ::i(2)                        [0]  int 0
↦ // ::i(1)                        [0]  int 0
↦ // ::work(1)                     [0]  string
↦ // ::varok                       [0]  string
```

6.5 Identifier resolution

In Singular, an identifier (i.e., a "word") is resolved in the following way and order: It is checked for

1. a reserved name (like `ring`, `std`, . . .),
2. a local variable (w.r.t. a procedure),
3. a local ring variable (w.r.t. the current basering locally set in a procedure),
4. a global ring variable (w.r.t. the current basering)
5. a global variable,
6. a monomial consisting of local ring variables written without operators,
7. a monomial consisting of global ring variables written without operators.

Consequently, it is allowed to have general variables with the same name as ring variables. However, the above identifier resolution order must be kept in mind. Otherwise, surprising results may come up.

```
ring r=0,(x,y),dp;
int x;
x*y; // resolved product int*poly, i.e., 0*y
↦ 0
xy; // "xy" is one identifier and resolved to monomial xy
↦ xy
```

For these reasons, we strongly recommend not to use variables which have the same name(s) as ring variables.

Moroever, we strongly recommend not to use ring variables whose name is fully contained in (i.e., is a substring of) another name of a ring variable. Otherwise, effects like the following might occur:

```
ring r=0,(x, x1),dp; // name x is substring of name x1 !!!!!!!!!
x;x1;   // resolved polynomial x
↦ x
↦ x1
short=0; 2x1; // resolved to monomial 2*x^1 !!!!!!
↦ 2*x
2*x1; // resolved to product 2 times x1
↦ 2*x1
```

7 Non-commutative subsystem

Singular has three non-commutative subsystems, handling various classes of associative non-commutative rings: Plural, Sca and Letterplace.

7.1 PLURAL

What is and what does Plural?

Plural is a kernel extension of Singular, providing many algorithms for computations within non-commutative $G-$ and $GR-$ algebras (see Section 7.4 [Mathematical background (plural)], page 365 for detailed information on algebras and algorithms).

It uses the same data structures as Singular, sometimes interpreting them in a different way and/or modifying them for its own purposes. In spite of such a difference, one can always transfer objects between commutative rings of Singular and non-commutative rings of Plural.

With Plural, one can set up a non-commutative G -algebra, say A , with a Poincaré-Birkhoff-Witt (PBW) basis, (see Section 7.4.1 [G-algebras], page 365 for step-by-step building instructions and also Section 7.5 [PLURAL libraries], page 370 for procedures for setting many important algebras easily). Afterwards, one can proceed to the factor-algebra of A modulo a two-sided ideal (see Section 7.3.29 [twostd (plural)], page 363), thus obtaining a GR -algebra (see Section 7.2.5 [qring (plural)], page 329 type).

Functionalities of Plural (enlisted in Section 7.3 [Functions (plural)], page 334) are accessible as soon as the basering becomes non-commutative (see Section 7.3.16 [nc_algebra], page 348 and the library Section 7.5.10 [ncalg_lib], page 464 with many readily predefined algebras).

One can perform various computations with polynomials and ideals in A and with vectors and submodules of a free module A^n.

What Plural does not:

Plural does not perform computations in the free algebra or in its general factor algebras (instead, these computations can be possibly done by Section 7.7 [LETTERPLACE], page 616).

In Plural one can only work with G -algebras and with their factor-algebras by two-sided ideals (GR -algebras).

Plural requires a global monomial ordering (see Section B.2.2 [General definitions for orderings], page 768). However, Sca (Section 7.6 [Graded commutative algebras (SCA)], page 614) provides the possibility of computations in a tensor product of a non-commutative graded commutative algebra (equipped with a global ordering) with a commutative algebra (equipped with any ordering).

Plural does not handle non-commutative parameters, i.e. the elements of the coefficient field (or a ring) mutually commute with all variables. Defining parameters, one **cannot** impose non-commutative relations on them. Moreover, it is impossible to introduce parameters which do not commute with variables. However, Section 7.5.21 [olga_lib], page 581 offers a rich functionality for working within Ore localizations of G -algebras and Section 7.5.25 [ratgb_lib], page 612 provides Groebner bases for so-called rational localizations of G -algebras.

Plural does not yet support rings like Z as coefficients.

Plural conventions

***-multiplication (plural)**

in the non-commutative case, the correct multiplication of `y` by `x` must be written as `y*x`.

Both expressions `yx` and `xy` are equal, since they are interpreted as commutative expressions. See example in Section 7.2.4.2 [poly expressions (plural)], page 328.

Note, that PLURAL output consists only of standard monomials, even when the signs `*` are omitted.

`ideal` **(plural)**

Unless stated otherwise, an expression of type `ideal` as understood by PLURAL as a list of generators of a **left** ideal. For more information see Section 7.2.1 [ideal (plural)], page 318.

For a **two-sided ideal** `T`, use the command Section 7.3.29 [twostd (plural)], page 363 for computing the two-sided Groebner basis of `T`.

For a **right ideal** `I`, use Section 7.8.10 [rightstd (letterplace)], page 632 from `nctools_lib` for computing the right Groebner basis of `I`.

`module` **(plural)**

Unless stated otherwise, a `module` as understood by PLURAL is **either** a finitely generated **left** submodule of a free module (of finite rank)

or a factor module of a free module (of finite rank) by its left submodule (see Section 7.2.3 [module (plural)], page 325 for details). The concrete interpretation left to a function.

`qring` **(plural)**

It is only possible to build factor-algebras modulo **two-sided** ideals (see Section 7.2.5 [qring (plural)], page 329), which have to be given via their two-sided Groebner basis (see Section 7.3.29 [twostd (plural)], page 363).

7.2 Data types (plural)

This chapter explains all data types of PLURAL in alphabetical order. For every type, there is a description of the declaration syntax
as well as information about how to build expressions of certain types.

The term "expression list" in PLURAL refers to any comma separated list of expressions.

For the general syntax of a declaration see Section 3.5.1 [General command syntax], page 41.

7.2.1 ideal (plural)

For PLURAL ideals are **left** ideals, unless stated otherwise.
Ideals are represented as lists of polynomials which are interpreted as left generators of the ideal.
For the operations with two-sided ideals see Section 7.3.29 [twostd (plural)], page 363.

Like polynomials, ideals can only be defined or accessed with respect to a basering.

Note: `size` counts only the non-zero generators of an ideal whereas `ncols` counts all generators.

7.2.1.1 ideal declarations (plural)

Syntax: `ideal` name = list_of_poly_and_ideal_expressions ;
`ideal` name = ideal_expression ;

Purpose: defines a left ideal.

Default: 0

Example:

```
ring r=0,(x,y,z),dp;
def R=nc_algebra(-1,0); // an anti-commutative algebra
setring R;
poly s1 = x2;
poly s2 = y3;
poly s3 = z;
ideal i =  s1, s2-s1, 0,s3*s2, s3^4;
i;
↦ i[1]=x2
↦ i[2]=y3-x2
↦ i[3]=0
↦ i[4]=-y3z
↦ i[5]=z4
size(i);
↦ 4
ncols(i);
↦ 5
```

7.2.1.2 ideal expressions (plural)

An ideal expression is:

1. an identifier of type ideal
2. a function returning an ideal
3. a combination of ideal expressions by the arithmetic operations + or *
4. a power of an ideal expression (operator ^ or **)
   Note that the computation of the product i*i involves all products of generators of i while i^2 involves only the different ones, and is therefore faster.
5. a type cast to ideal

Example:

```
ring r=0,(x,y,z),dp;
def R=nc_algebra(-1,0); // an anticommutative algebra
setring R;
ideal m = maxideal(1);
m;
↦ m[1]=x
↦ m[2]=y
↦ m[3]=z
poly f = x2;
poly g = y3;
ideal i = x*y*z , f-g, g*(x-y) + f^4 ,0, 2x-z2y;
ideal M = i + maxideal(10);
i = M*M;
ncols(i);
↦ 598
i = M^2;
ncols(i);
↦ 690
i[ncols(i)];
↦ x20
```

```
vector v = [x,y-z,x2,y-x,x2yz2-y];
ideal j = ideal(v);
j;
↦ j[1]=x
↦ j[2]=y-z
↦ j[3]=x2
↦ j[4]=-x+y
↦ j[5]=x2yz2-y
```

7.2.1.3 ideal operations (plural)

`+` addition (concatenation of the generators and simplification)

`*` multiplication (with ideal, poly, vector, module; in case of multiplication with ideal or module, the result will be simplified)

`^` exponentiation (by a non-negative integer)

ideal_expression [intvec_expression]
are polynomial generators of the ideal, index 1 gives the first generator.

Note: For simplification of an ideal, see also Section 5.1.143 [simplify], page 263.

Example:

```
ring r=0,(x,y,z),dp;
matrix D[3][3];
D[1,2]=-z;  D[1,3]=y;  D[2,3]=x;
def R=nc_algebra(1,D); // this algebra is U(so_3)
setring R;
ideal I = 0,x,0,1;
I;
↦ I[1]=0
↦ I[2]=x
↦ I[3]=0
↦ I[4]=1
I + 0;    // simplification
↦ _[1]=1
I*x;
↦ _[1]=0
↦ _[2]=x2
↦ _[3]=0
↦ _[4]=x
ideal J = I,0,x,x-z;
I * J;   //  multiplication with simplification
↦ _[1]=1
vector V = [x,y,z];
print(I*V);
↦ 0,x2,0,x,
↦ 0,xy,0,y,
↦ 0,xz,0,z
ideal m = maxideal(1);
m^2;
↦ _[1]=x2
↦ _[2]=xy
↦ _[3]=xz
↦ _[4]=y2
↦ _[5]=yz
↦ _[6]=z2
ideal II = I[2..4];
II;
↦ II[1]=x
↦ II[2]=0
↦ II[3]=1
```

7.2.1.4 ideal related functions (plural)

`dim` Gelfand-Kirillov dimension of basering modulo the ideal of leading terms (see Section 7.3.3 [dim (plural)], page 336)

`eliminate` elimination of variables (see Section 7.3.5 [eliminate (plural)], page 338)

`intersect` ideal intersection (see Section 7.3.9 [intersect (plural)], page 342)

`kbase` vector space basis of basering modulo the leading ideal (see Section 7.3.10 [kbase (plural)], page 342)

`lead` leading terms of a set of generators (see Section 5.1.75 [lead], page 209)

`lift` lift-matrix (see Section 7.3.11 [lift (plural)], page 343)

`liftstd` left Groebner basis and transformation matrix computation (see Section 7.3.12 [liftstd (plural)], page 344)

`maxideal` generators of a power of the maximal ideal at 0 (see Section 5.1.88 [maxideal], page 219)

`modulo` represents $(h1+h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 7.3.14 [modulo (plural)], page 346)

`mres` minimal free resolution of an ideal and a minimal set of generators of the given ideal (see Section 7.3.15 [mres (plural)], page 347)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`nres` computes a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 7.3.18 [nres (plural)], page 350)

`oppose` creates an opposite ideal of a given ideal from the given ring into a basering (see Section 7.3.19 [oppose], page 352)

`preimage` preimage under a ring map (see Section 7.3.21 [preimage (plural)], page 354)

`quotient` ideal quotient (see Section 7.3.22 [quotient (plural)], page 355)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 356)

`simplify` simplify a set of polynomials (see Section 5.1.143 [simplify], page 263)

`size` number of non-zero generators (see Section 5.1.144 [size], page 264)

`slimgb` left Groebner basis computation with slim technique (see Section 7.3.25 [slimgb (plural)], page 359)

`std` left Groebner basis computation (see Section 7.3.26 [std (plural)], page 360) `subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 362)

`syz` computation of the first syzygy module (see Section 7.3.28 [syz (plural)], page 362)

`twostd` two-sided Groebner basis computation (see Section 7.3.29 [twostd (plural)], page 363)

`vdim` vector space dimension of basering modulo the leading ideal (see Section 7.3.30 [vdim (plural)], page 364)

7.2.2 map (plural)

Maps are ring maps from a preimage ring (source) into the basering (target), defined by specifying images for source variables in the target ring.

Note:

- the target of a map is **ALWAYS** the actual basering
- the preimage ring has to be stored "by its name", that means, maps can only be used in such contexts, where the name of the preimage ring can be resolved (this has to be considered in subprocedures). See also Section 6.5 [Identifier resolution], page 315, Section 3.7.4 [Names in procedures], page 54.

Maps between rings with different coefficient fields are possible and listed below.

Canonically realized are

- $Q \to Q(a,\ldots)$ (Q : the rational numbers)
- $Q \to R$ (R : the real numbers)
- $Q \to C$ (C : the complex numbers)
- $Z/p \to (Z/p)(a,\ldots)$ (Z : the integers)
- $Z/p \to GF(p^n)$ (GF : the Galois field)
- $Z/p \to R$
- $R \to C$

Possible are furthermore

- $Z/p \to Q, \quad [i]_p \mapsto i \in [-p/2,\, p/2] \subseteq Z$
- $Z/p \to Z/p', \quad [i]_p \mapsto i \in [-p/2,\, p/2] \subseteq Z,\ i \mapsto [i]_{p'} \in Z/p'$
- $C \to R, \quad$ by taking the real part

Finally, in Plural we allow the mapping from rings with coefficient field Q to rings whose ground fields have finite characteristic:

- $Q \to Z/p$
- $Q \to (Z/p)(a,\ldots)$

Note: In these cases the denominator and the numerator of a number are mapped separately by the usual map from Z to Z/p, and the image of the number is built again afterwards by division. It is thus not allowed to map numbers whose denominator is divisible by the characteristic of the target ground field, or objects containing such numbers. We, therefore, strongly recommend to use such maps only to map objects with integer coefficients.
Note that - in contrast to the commutative case - maps between non-commutative rings easily fail to be a morphism.

7.2.2.1 map declarations (plural)

Syntax: `map` name = preimage_ring_name , ideal_expression ;
`map` name = preimage_ring_name , list_of_poly_and_ideal_expressions ;
`map` name = map_expression ;

Purpose: defines a ring map from `preimage_ring` to basering.
Maps the variables of the `preimage ring` to the generators of the ideal.
If the ideal contains less elements than the number of variables in the `preimage_ring`, the remaining variables are mapped to 0.
If the ideal contains more elements, extra elements are ignored.
The image ring is always the current basering. For the mapping of coefficients from different fields see Section 7.2.2 [map (plural)], page 322.

Default: none

Note: There are standard mappings for maps which are close to the identity map: `fetch (plural)` and `imap (plural)`.

The name of a map serves as the function which maps objects from the preimage_ring into the basering. These objects must be defined by names (no evaluation in the preimage ring is possible).

Example:

```
// an easy example
ring r1 = 0,(a,b),dp; // a commutative ring
poly P = a^2+ab+b^3;
ring r2 = 0,(x,y),dp;
def W=nc_algebra(1,-1); // a Weyl algebra
setring W;
map M = r1, x^2, -y^3;
// note: M is just a map and not a morphism of K-algebras
M(P);
↦ -y9-x2y3+x4
// now, a more involved example
LIB "ncalg.lib";
def Usl2 = makeUsl2();
// this algebra is U(sl_2), generated by  e,f,h
setring Usl2;
poly  P  = 4*e*f+h^2-2*h; // the central el-t of Usl2
poly  Q  = e^3*f-h^4;     // some polynomial
ring W1  = 0,(D,X),dp;
def W2=nc_algebra(1,-1);
setring W2; // this is the opposite Weyl algebra
map F = Usl2, -X, D*D*X, 2*D*X;
F(P); // 0, because P is in the kernel of F
↦ 0
F(Q);
↦ -16D4X4+96D3X3-D2X4-112D2X2+6DX3+16DX-6X2
```

See Section 7.3.7 [fetch (plural)], page 340; Section 7.2.1.2 [ideal expressions (plural)], page 319; Section 7.3.8 [imap (plural)], page 341; Section 7.2.2 [map (plural)], page 322; Section 7.2.7 [ring (plural)], page 332.

7.2.2.2 map expressions (plural)

A map expression is:

1. an identifier of type map
2. a function returning map
3. a composition of maps using parentheses, e.g. `f(g)`

7.2.2.3 map (plural) operations

`( )`
: composition of maps. If, for example, `f` and `g` are maps, then `f(g)` is a map expression giving the composition $f \circ g$ of `f` and `g`, provided the target ring of `g` is the basering of `f`.

map_expression `[` int_expressions `]`
: is a map entry (the image of the corresponding variable)

Example:

```
LIB "ncalg.lib";
def Usl2 = makeUsl2(); // this algebra is U(sl_2)
setring Usl2;
map F = Usl2, f, e, -h; // involutive endomorphism of U(sl_2)
F;
↦ F[1]=f
↦ F[2]=e
↦ F[3]=-h
map G = F(F);
G;
↦ G[1]=e
↦ G[2]=f
↦ G[3]=h
poly p = (f+e*h)^2 + 3*h-e;
p;
↦ e2h2+2e2h+2efh-2ef+f2-h2-e+3h
F(p);
↦ f2h2-2efh-2f2h+e2-2ef+h2-f-h
G(p);
↦ e2h2+2e2h+2efh-2ef+f2-h2-e+3h
(G(p) == p); // G is the identity
↦ 1
```

7.2.2.4 map related functions (plural)

`fetch (plural)`
: the identity map between rings and qrings (see Section 7.3.7 [fetch (plural)], page 340)

`imap (plural)`
: a convenient map procedure for inclusions and projections of rings (see Section 7.3.8 [imap (plural)], page 341)

`preimage (plural)`
: preimage under a ring map (see Section 7.3.21 [preimage (plural)], page 354)

`subst`
: substitute a ring variable (see Section 7.3.27 [subst (plural)], page 362)

See also Section 7.5.19 [ncpreim_lib], page 554 for the advanced preimage algorithm.

7.2.3 module (plural)

Modules are **left** submodules of a free module over the basering with basis gen(1), gen(2), ..., gen(n) for some natural number n.
They are represented by lists of vectors, which generate the left submodule. Like vectors, they can only be defined or accessed with respect to a basering.

If M is a left submodule of R^n (where R is the basering) generated by vectors $v_1,\ldots,v_k$, then these generators may be considered as the generators of relations of R^n/M between the canonical generators gen(1),...,gen(n). Hence, any finitely generated R -module can be represented in Plural by its module of relations. This is the so-called Coker-representation.
The assignments module M=v1,...,vk; matrix A=M; create the presentation matrix of size $n\times k$, with the columns of A being the vectors $v_1,\ldots,v_k$ which generate M.

7.2.3.1 module declarations (plural)

Syntax: module name = list_of_vector_expressions (which are interpreted as left generators of the module) ;
module name = module_expression ;

Purpose: defines a left module.

Default: [0]

Example:

```
ring r=0,(x,y,z),(c,dp);
matrix D[3][3];
D[1,2]=-z;  D[1,3]=y;  D[2,3]=x;
def R=nc_algebra(1,D); // this algebra is U(so_3)
setring R;
vector s1 = [x2,y3,z];
vector s2 = [xy,1,0];
vector s3 = [0,x2-y2,z];
poly   f  = -x*y;
module m = s1, s2-s1,f*(s3-s1);
m;
↦ m[1]=[x2,y3,z]
↦ m[2]=[-x2+xy,-y3+1,-z]
↦ m[3]=[x3y-2x2z-xy,xy4-x3y+xy3+2x2z+xy]
// show m in matrix format (columns generate m)
print(m);
↦ x2,-x2+xy,x3y-2x2z-xy,
↦ y3,-y3+1, xy4-x3y+xy3+2x2z+xy,
↦ z, -z,    0
```

7.2.3.2 module expressions (plural)

A module expression is:

1. an identifier of type module
2. a function returning module
3. module expressions combined by the arithmetic operation +
4. multiplication of a module expression with an ideal or a poly expression: *
5. a type cast to module

7.2.3.3 module operations (plural)

+ addition (concatenation of the generators and simplification) Note that "-" implicitly converts a module into a matrix; see below example.

* right or left multiplication with number, ideal, or poly (but not ‘module‘ * ‘module‘!)

module_expression [int_expression , int_expression]
is a module entry, where the first index indicates the row and the second the column

module_expressions [int_expression]
is a vector, where the index indicates the column (generator)

Example:

```
ring A=0,(x,y,z),Dp;
matrix D[3][3];
D[1,2]=-z;  D[1,3]=y;  D[2,3]=x;  // this algebra is U(so_3)
def B=nc_algebra(1,D);
setring B;
module M = [x,y],[0,0,x*z];
module N = matrix((x+y-z)*M) - matrix(M*(x+y-z)); // no - for type module
print(N);
↦ -y-z,0,
↦ -x+z,0,
↦ 0,   -x2-xy-yz-z2
```

7.2.3.4 module related functions (plural)

`eliminate`
elimination of variables (see Section 7.3.5 [eliminate (plural)], page 338)

`freemodule`
the free module of given rank (see Section 5.1.47 [freemodule], page 188)

`intersect`
module intersection (see Section 7.3.9 [intersect (plural)], page 342)

`kbase` vector space basis of free module over the basering modulo the module of leading terms (see Section 7.3.10 [kbase (plural)], page 342)

`lead` initial module (see Section 5.1.75 [lead], page 209)

`lift` lift-matrix (see Section 7.3.11 [lift (plural)], page 343)

`liftstd` left Groebner basis and transformation matrix computation (see Section 7.3.12 [liftstd (plural)], page 344)

`modulo` represents $(h1+h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 7.3.14 [modulo (plural)], page 346)

`mres` minimal free resolution of a module and a minimal set of generators of the given ideal module (see Section 7.3.15 [mres (plural)], page 347)

`ncols` number of columns (see Section 5.1.104 [ncols], page 231)

`nres` computes a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 7.3.18 [nres (plural)], page 350)

`nrows` number of rows (see Section 5.1.107 [nrows], page 232) `oppose` creates an opposite module of a given module from the given ring into a basering (see Section 7.3.19 [oppose], page 352)

`print` nice print format (see Section 5.1.120 [print], page 242)

`prune` minimize the embedding into a free module (see Section 5.1.122 [prune], page 245)

`quotient` module quotient (see Section 7.3.22 [quotient (plural)], page 355)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 356)

`simplify` simplify a set of vectors (see Section 5.1.143 [simplify], page 263)

`size` number of non-zero generators (see Section 5.1.144 [size], page 264)

`std` left Groebner basis computation (see Section 7.3.26 [std (plural)], page 360)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 362)

`syz` computation of the first syzygy module (see Section 7.3.28 [syz (plural)], page 362)

`vdim` vector space dimension of free module over the basering modulo module of leading terms (see Section 7.3.30 [vdim (plural)], page 364)

7.2.4 poly (plural)

Polynomials and vectors are the basic data for all main algorithms in Plural. Polynomials consist of finitely many terms (coefficient*monomial) which are combined by the usual polynomial operations (see Section 7.2.4.2 [poly expressions (plural)], page 328). Polynomials can only be defined or accessed with respect to a basering which determines the coefficient type, the names of the indeterminants and the monomial ordering.

Example:

```
ring r=32003,(x,y,z),dp;
poly f=x3+y5+z2;
```

Remark: Remember the conventions on polynomial multiplication we follow (*-multiplication in Section 7.1 [PLURAL], page 317).

7.2.4.1 poly declarations (plural)

Syntax: `poly` name = poly_expression `;`

Purpose: defines a polynomial.

Default: 0

Example:

```
ring r = 32003,(x,y,z),dp;
def R=nc_algebra(-1,1);
setring R;
// ring of some differential-like operators
R;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
```

```
↦ //    yx=-xy+1
↦ //    zx=-xz+1
↦ //    zy=-yz+1
yx;      // not correct input
↦ xy
y*x;     // correct input
↦ -xy+1
poly s1  = x3y2+151x5y+186xy6+169y9;
poly s2  = 1*x^2*y^2*z^2+3z8;
poly s3  = 5/4x4y2+4/5*x*y^5+2x2y2z3+y7+11x10;
int a,b,c,t=37,5,4,1;
poly f=3*x^a+x*y^(b+c)+t*x^a*y^b*z^c;
f;
↦ x37y5z4+3x37+xy9
short = 0;
f;
↦ x^37*y^5*z^4+3*x^37+x*y^9
```

7.2.4.2 poly expressions (plural)

A polynomial expression is (optional parts in square brackets):

1. a monomial (there are NO spaces allowed inside a monomial)

   `[coefficient] ring_variable [exponent] [ring_variable [exponent] ...]`

   monomials which contain an indexed ring variable must be built from `ring_variable` and `coefficient` with the operations `*` and `^`
2. an identifier of type poly
3. a function returning poly
4. polynomial expressions combined by the arithmetic operations `+`, `-`, `*`, `/`, or `^`.
5. a type cast to poly

Example:

```
ring r=0,(x,y),dp;
def R=nc_algebra(1,1);  // make it a Weyl algebra
setring R;
R;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=xy+1
yx;      // not correct input
↦ xy
y*x;     // correct input
↦ xy+1
poly f = 10x2*y3 + 2y2*x^2 - 2*x*y + y - x + 2;
lead(f);
↦ 10x2y3
leadmonom(f);
```

```
↦ x2y3
simplify(f,1);     // normalize leading coefficient
↦ x2y3+1/5x2y2+3/5xy-1/10x+1/10y+3/5
cleardenom(f);
↦ 10x2y3+2x2y2+6xy-x+y+6
```

7.2.4.3 poly operations (plural)

`+` addition

`-` negation or subtraction

`*` multiplication

`/` commutative division by a monomial, non divisible terms yield 0

`^`, `**` power by a positive integer

`<`, `<=`, `>`, `>=`, `==`, `<>`
comparison (of leading monomials w.r.t. monomial ordering)

poly_expression `[` intvec_expression `]`
the sum of monomials at the indicated places w.r.t. the monomial ordering

7.2.4.4 poly related functions (plural)

`bracket` computes the (iterated) Lie bracket of two polynomials (see Section 7.3.2 [bracket], page 335)

`lead` leading term (see Section 5.1.75 [lead], page 209)

`leadcoef` coefficient of the leading term (see Section 5.1.76 [leadcoef], page 209)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.77 [leadexp], page 210)

`leadmonom`
leading monomial (see Section 5.1.78 [leadmonom], page 210)

`oppose` creates an opposite polynomial of a given polynomial from the given ring into a basering (see Section 7.3.19 [oppose], page 352)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 356)

`simplify` normalize a polynomial (see Section 5.1.143 [simplify], page 263)

`size` number of monomials (see Section 5.1.144 [size], page 264)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 362)

`var` the indicated variable of the ring (see Section 5.1.165 [var], page 285)

7.2.5 qring (plural)

Plural offers the possibility to compute within factor-rings modulo two-sided ideals. The ideal has to be given as a two-sided Groebner basis (see Section 7.3.29 [twostd (plural)], page 363 command).

For a detailed description of the concept of rings and quotient rings see Section 3.3 [Rings and orderings], page 30.

Note: we highly recommend to turn on `option(redSB); option(redTail);` while computing in qrings. Otherwise results may have a difficult interpretation.

7.2.5.1 qring declaration (plural)

Syntax: `qring` name = ideal_expression ;

Default: none

Purpose: declares a quotient ring as the basering modulo an `ideal_expression` and sets it as current basering.

Note: reports error if an ideal is not a two-sided Groebner basis.

Example:

```
ring r=0,(z,u,v,w),dp;
def R=nc_algebra(-1,0); // an anticommutative algebra
setring R;
option(redSB);
option(redTail);
ideal i=z^2,u^2,v^2,w^2, zuv-w;
qring Q = i;  // incorrect call produces error
↦ // ** i is no standard basis
↦ // ** i is no twosided standard basis
kill Q;
setring R;  // go back to the ring R
qring q=twostd(i); // now it is an exterior algebra modulo <zuv-w>
q;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    z u v w
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    uz=-zu
↦ //    vz=-zv
↦ //    wz=-zw
↦ //    vu=-uv
↦ //    wu=-uw
↦ //    wv=-vw
↦ // quotient ring from ideal
↦ _[1]=w2
↦ _[2]=vw
↦ _[3]=uw
↦ _[4]=zw
↦ _[5]=v2
↦ _[6]=u2
↦ _[7]=z2
↦ _[8]=zuv-w
poly k = (v-u)*(zv+u-w);
k; // the output is not yet totally reduced
↦ zuv-uv+uw-vw
poly ek=reduce(k,std(0));
ek; // the reduced form
↦ -uv+w
```

7.2.5.2 qring related functions (plural)

`envelope` enveloping ring (see Section 7.3.6 [envelope], page 339)

`nvars` number of ring variables (see Section 5.1.109 [nvars], page 233)

`opposite` opposite ring (see Section 7.3.20 [opposite], page 353)

`setring` set a new basering (see Section 5.1.141 [setring], page 260)

7.2.6 resolution (plural)

The type resolution is intended as an intermediate representation which internally retains additional information obtained during computation of resolutions. It furthermore enables the use of partial results to compute, for example, Betti numbers or minimal resolutions. Like ideals and modules, a resolution can only be defined w.r.t. a basering.

Note: to access the elements of a resolution, it has to be assigned to a list. This assignment also completes computations and may therefore take time, (resp. an access directly with the brackets `[ , ]` causes implicitly a cast to a list).

7.2.6.1 resolution declarations (plural)

Syntax: `resolution` name = resolution_expression ;

Purpose: defines a resolution.

Default: none

Example:

```
ring r=0,(x,y,z),dp;
matrix D[3][3];
D[1,2]=z;
def R=nc_algebra(1,D); // it is a Heisenberg algebra
setring R;
ideal i=z2+z,x+y;
resolution re=nres(i,0);
re;
↦  1      2      1
↦ R <--  R <--  R
↦
↦ 0      1      2
↦ resolution not minimized yet
↦
list l = re;
l;
↦ [1]:
↦    _[1]=z2+z
↦    _[2]=x+y
↦ [2]:
↦    _[1]=z2*gen(2)-x*gen(1)-y*gen(1)+z*gen(2)
↦ [3]:
↦    _[1]=0
print(matrix(l[2]));
↦ -x-y,
↦ z2+z
print(module(transpose(matrix(l[2]))*transpose(matrix(l[1]))));  // check
↦ 0
```

7.2.6.2 resolution expressions (plural)

A resolution expression is:

1. an identifier of type resolution
2. a function returning a resolution
3. a type cast to resolution from a list of ideals, resp. modules.

7.2.6.3 resolution related functions (plural)

`betti` Betti numbers of a resolution (see Section 7.3.1 [betti (plural)], page 334)

`minres` minimizes a free resolution (see Section 7.3.13 [minres (plural)], page 345)

`mres` computes a minimal free resolution of an ideal resp. module and a minimal set of generators of the given ideal resp. module (see Section 7.3.15 [mres (plural)], page 347)

`nres` computes a free resolution of an ideal resp. module M which is minimized from the second module on (see Section 7.3.18 [nres (plural)], page 350)

7.2.7 ring (plural)

Rings are used to describe properties of polynomials, ideals etc. Almost all computations in Plural require a basering. For a detailed description of the concept of rings see Section 3.3 [Rings and orderings], page 30.

Note: Plural usually works with global orderings (see Section 7.1 [PLURAL], page 317) but one can use certain local once when graded commutative rings are being used.

7.2.7.1 ring declarations (plural)

Syntax: `ring` name `=` `(` coefficient_field `)`, `(` names_of_ring_variables `)`, `(` ordering `)`;

Default: `32003,(x,y,z),(dp,C);`

Purpose: declares a ring and sets it as the actual basering.

The coefficient_field is given by one of the following:

1. a non-negative int_expression less or equal 2147483647.
2. an expression_list of an int_expression and one or more names.
3. the name `real`.
4. an expression_list of the name `real` and an int_expression.
5. an expression_list of the name `complex`, an optional int_expression and a name.

'names_of_ring_variables' must be a list of names or indexed names.

'ordering' is a list of block orderings where each block ordering is either

1. `lp`, `dp`, `Dp`, optionally followed by a size parameter in parentheses.
2. `wp`, `Wp`, or `a` followed by a weight vector given as an intvec_expression in parentheses.
3. `M` followed by an intmat_expression in parentheses.
4. `c` or `C`.

As long as all non-commuting variables are global, any ordering may be used. In graded commutative algebras, one may also use `ls, ds, Ds, ws`, and `Ws`.

If one of coefficient_field, names_of_ring_variables, and ordering consists of only one entry, the parentheses around this entry may be omitted.

In order to create a non-commutative structure over a commutative ring, use Section 7.3.16 [nc_algebra], page 348.

7.2.7.2 ring operations (plural)

`+` construct a tensor product $C = A \otimes_{\mathbf{K}} B$ of two G-algebras A and B over the ground field. Let, e.g.,

$A = k_1\langle x_1, \ldots, x_n \mid \{x_j x_i = c_{ij} \cdot x_i x_j + d_{ij}\}, 1 \leq i < j \leq n\rangle$, and $B = k_2\langle y_1, \ldots, y_m \mid \{y_j y_i = q_{ij} \cdot y_i y_j + r_{ij}\}, 1 \leq i < j \leq m\rangle$

be two G -algebras, then C is defined to be the algebra

$C = K\langle x_1, \ldots, x_n, y_1, \ldots, y_m \mid \{x_j x_i = c_{ij} \cdot x_i x_j + d_{ij}, 1 \leq i < j \leq n\}, \{y_j y_i = q_{ij} \cdot y_i y_j + r_{ij}, 1 \leq i < j \leq m\}, \{y_j x_i = x_i y_j, 1 \leq j \leq m, 1 \leq i \leq n\}\rangle$.

Concerning the ground fields k_1 resp. k_2 of A resp. B , take the following guidelines for $A \otimes_{\mathbf{K}} B$ into consideration:

- Neither k_1 nor k_2 may be R or C .
- If the characteristic of k_1 and k_2 differs, then one of them must be Q .
- At most one of k_1 and k_2 may have parameters.
- If one of k_1 and k_2 is an algebraic extension of Z/p it may not be defined by a `charstr` of type `(p^n,a)`.

One can create a ring using `ring(list)`, see also `ringlist`.

Example:

```
LIB "ncalg.lib";
def a = makeUsl2();       // U(sl_2) in e,f,h presentation
ring W0 = 0,(x,d),dp;
def W = Weyl();              // 1st Weyl algebra in x,d
def S = a+W;
setring S;
S;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering dp
↦ //                  : names    x d
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
↦ //    dx=xd+1
```

7.2.7.3 ring related functions (plural)

`charstr` description of the coefficient field of a ring (see Section 5.1.7 [charstr], page 162) `envelope` enveloping ring (see Section 7.3.6 [envelope], page 339)

`npars` number of ring parameters (see Section 5.1.105 [npars], page 231)

`nvars` number of ring variables (see Section 5.1.109 [nvars], page 233)

`opposite` opposite ring (see Section 7.3.20 [opposite], page 353)

`ordstr` monomial ordering of a ring (see Section 5.1.113 [ordstr], page 239)

`parstr` names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.116 [parstr], page 240)

`qring` quotient ring (see Section 7.2.5 [qring (plural)], page 329)

`ringlist` decomposes a ring into a list of its components (see Section 7.3.24 [ringlist (plural)], page 357)

`setring` set a new basering (see Section 5.1.141 [setring], page 260)

`varstr` names of all ring variables or the name of the n-th ring variable (see Section 5.1.167 [varstr], page 285)

7.3 Functions (plural)

This chapter gives a complete reference of all functions and commands of the PLURAL kernel, i.e. all built-in commands (for the PLURAL libraries see Section 7.5 [PLURAL libraries], page 370).

The general syntax of a function is

[target =] function_name (<arguments>);

Note, that both **Control structures** and **System variables** of PLURAL are the same as of SINGULAR (see Section 5.2 [Control structures], page 290, Section 5.3 [System variables], page 302).

7.3.1 betti (plural)

Syntax: `betti` (list_expression)
`betti` (resolution_expression)
`betti` (list_expression , int_expression)
`betti` (resolution_expression , int_expression)

Type: intmat

Note: in the non-commutative case, computing Betti numbers makes sense only if the basering R has homogeneous relations. The output of the command can be pretty-printed using `print( , ''betti'')`, i.e., with "betti" as second argument; see below example.

Purpose: with 1 argument: computes the graded Betti numbers of a minimal resolution of R^n/M, if R denotes the basering and M a homogeneous submodule of R^n and the argument represents a resolution of R^n/M.
The entry d of the intmat at place (i, j) is the minimal number of generators in degree i+j of the j-th syzygy module (= module of relations) of R^n/M (the 0th (resp. 1st) syzygy module of R^n/M is R^n (resp. M)). The argument is considered to be the result of a `mres` or `nres` command. This implies that a zero is only allowed (and counted) as a generator in the first module.
For the computation `betti` uses only the initial monomials. This could lead to confusing results for a non-homogeneous input.

If the optional second argument is non-zero, the Betti numbers will be minimized.

Example:

```
int i;int N=2;
ring r=0,(x(1..N),d(1..N),q(1..N)),Dp;
matrix D[3*N][3*N];
for (i=1;i<=N;i++)
{ D[i,N+i]=q(i)^2; }
def W=nc_algebra(1,D); setring W;
// this algebra is a kind of homogenized Weyl algebra
W;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering Dp
↦ //                  : names    x(1) x(2) d(1) d(2) q(1) q(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    d(1)x(1)=x(1)*d(1)+q(1)^2
↦ //    d(2)x(2)=x(2)*d(2)+q(2)^2
ideal I = x(1),x(2),d(1),d(2),q(1),q(2);
option(redSB);
option(redTail);
resolution R = mres(I,0);
// thus R will be the full length minimal resolution
print(betti(R),"betti");
↦            0     1     2     3     4     5     6
↦ ------------------------------------------------
↦     0:     1     6    15    20    15     6     1
↦ ------------------------------------------------
↦ total:     1     6    15    20    15     6     1
↦
```

7.3.2 bracket

Syntax: `bracket (` poly_expression, poly_expression `)`
`bracket (` poly_expression, poly_expression, int_expression `)`

Type: poly

Purpose: Computes the Lie bracket `[p,q]=pq-qp` of the first polynomial with the second. Uses special routines, based on the Leibniz rule. If the third argument is `N>1`, then the right normed bracket `[a,[...[a,b]]]]` will be computed.

Note: effective both with Plural and Letterplace rings.

Example:

```
ring r=(0,Q),(x,y,z),Dp; // first, let us do a Plural example
minpoly=Q^2-Q+1;
matrix C[3][3];  matrix D[3][3];
C[1,2]=Q2;    C[1,3]=1/Q2;  C[2,3]=Q2;
D[1,2]=-Q*z;  D[1,3]=1/Q*y; D[2,3]=-Q*x;
def R=nc_algebra(C,D); setring R; R;
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering Dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
// this is a quantum deformation of U(so_3),
// where Q is a 6th root of unity
poly p=Q^4*x2+y2+Q^4*z2+Q*(1-Q^4)*x*y*z;
// p is the central element of the algebra
p=p^3; // any power of a central element is central
poly q=(x+Q*y+Q^2*z)^4;
// take q to be some big noncentral element
size(q); // check how many monomials are in big polynomial q
↦ 28
bracket(p,q); // check p*q=q*p
↦ 0
// a more common behaviour of the bracket follows:
bracket(x+Q*y+Q^2*z,z);
↦ (Q+1)*xz+(Q+1)*yz+(Q-1)*x+(Q-1)*y
kill R; setring r; // Now consider an example for Letterplace
LIB "freegb.lib";
ring R = freeAlgebra(r,5); // F<x,y,z> with deg left lex ordering
bracket(x,y);
↦ x*y-y*x
bracket(x,y,2);
↦ x*x*y-2*x*y*x+y*x*x
bracket(x,y,3);
↦ x*x*x*y-3*x*x*y*x+3*x*y*x*x-y*x*x*x
bracket(z^2,x+Q*y,2);
↦ x*z*z*z*z+(Q)*y*z*z*z*z-2*z*z*x*z*z+(-2*Q)*z*z*y*z*z+z*z*z*z*x+(Q)*z*z*z
   z*y
```

7.3.3 dim (plural)

Syntax: `dim` (ideal_expression)
`dim` (module_expression)

Type: int

Purpose: computes the Gelfand-Kirillov dimension of the ideal, resp. module, generated by the leading monomials of the given generators of the ideal, resp. module. This is also the dimension of the ideal resp. submodule, if it is represented by a left Groebner basis.

Note: The dimension of a submodule of a free module is defined to be the Gelfand-Kirillov dimension of the left module with the presentation via given submodule.
The computed Gelfand-Kirillov dimension is taken relative to the ground field. In order to compute the complete Gelfand-Kirillov dimension, one has to add the transcendence degree of the ground field over its prime field.

Example:

```
ring r=0,(x,y,Dx,Dy),dp;
matrix M[4][4]; M[1,3]=1;M[2,4]=1;
def R = nc_algebra(1,M); // 2nd Weyl algebra
setring R;
```

```
  dim(std(0)); // the GK dimension of the ring itself
↦ 4
  ideal I=x*Dy^2-2*y*Dy^2+2*Dy, Dx^3+3*Dy^2;
  dim(std(I)); // the GK dimension of the module R/I
↦ 2
  module T = (x*Dx -2)*gen(1), Dx^3*gen(1), (y*Dy +3)*gen(2);
  dim(std(T)); // the GK dimension of the module  R^2/T
↦ 3
```

See Section 4.6 [ideal], page 80; Section 4.14 [module], page 112; Section 5.1.151 [std], page 271; Section 5.1.168 [vdim], page 286.

7.3.4 division (plural)

Syntax: `division` (ideal_expression, ideal_expression)
`division` (module_expression, module_expression)
`division` (ideal_expression, ideal_expression, int_expression)
`division` (module_expression, module_expression, int_expression)
`division` (ideal_expression, ideal_expression, int_expression, intvec_expression)
`division` (module_expression, module_expression, int_expression, intvec_expression)

Type: list

Purpose: `division` computes a left division with remainder. For two left ideals resp. modules `M` (first argument) and `N` (second argument), it returns a list `T,R,U` where `T` is a matrix, `R` is a left ideal resp. a module, and `U` is a diagonal matrix of units such that `transpose(U)*transpose(matrix(M))=transpose(T)*transpose(matrix(N)) + transpose(matrix(R))`. From this data one gets a left standard representation for the left normal form `R` of `M` with respect to a left Groebner basis of `N`. `division` uses different algorithms depending on whether `N` is represented by a Groebner basis. For a GR-algebra, the matrix `U` is the identity matrix. A matrix `T` as above is also computed by `lift`.
For additional arguments `n` (third argument) and `w` (fourth argument), `division` returns a list `T,R` as above such that `transpose(matrix(M))=transpose(T)*transpose(matrix(N)) + transpose(matrix(R))` is a left standard representation for the left normal form `R` of `M` with respect to `N` up to weighted degree `n` with respect to the weight vector `w`. The weighted degree of `T` and `R` respect to `w` is at most `n`. If the weight vector `w` is not given, `division` uses the standard weight vector `w=1,...,1`.

Example:

```
  LIB "dmod.lib";
  ring r = 0,(x,y),dp;
  poly f = x^3+xy;
  def S = Sannfs(f); setring S; // compute the annihilator of f^s
  LD; // is not a Groebner basis yet!
↦ LD[1]=3*x^2*Dy-x*Dx+y*Dy
↦ LD[2]=x*Dx+2*y*Dy-3*s
  poly f = imap(r,f);
  poly P = f*Dx-s*diff(f,x);
  division(P,LD); // so P is in the ideal via the cofactors in _[1]
↦ [1]:
```

```
↦     _[1,1]=-2/3*y
↦     _[2,1]=x^2+1/3*y
↦  [2]:
↦     _[1]=0
↦  [3]:
↦     _[1,1]=1
ideal I = LD, f; // consider a bigger ideal
list L = division(s^2, I); // the normal form is -2s-1
L;
↦  [1]:
↦     _[1,1]=2/3*x^2*Dy-1/3*x*Dx+2/3*s+1/3
↦     _[2,1]=2/3*x^2*Dy-1/3*x*Dx-1/3*s-2/3
↦     _[3,1]=-2*x*Dy^2+Dx*Dy
↦  [2]:
↦     _[1]=-2*s-1
↦  [3]:
↦     _[1,1]=1
// now we show that the formula above holds
matrix M[1][1] = s^2; matrix N = matrix(I);
matrix T = matrix(L[1]); matrix R = matrix(L[2]); matrix U  = matrix(L[3])
// the formula must return zero:
transpose(U)*transpose(M) - transpose(T)*transpose(N) - transpose(R);
↦  _[1,1]=0
```

See Section 4.6 [ideal], page 80; Section 5.1.80 [lift], page 211; Section 4.14 [module], page 112; Section 4.17 [poly], page 120; Section 4.23 [vector], page 134.

7.3.5 eliminate (plural)

Syntax: `eliminate (` ideal_expression`,` product_of_ring_variables`)`
`eliminate (` module_expression`,` product_of_ring_variables`)`

Type: the same as the type of the first argument

Purpose: eliminates variables occurring as factors of the second argument from an ideal (resp. a submodule of a free module), by intersecting it (resp. each component of the submodule) with the subring not containing these variables.

Note: `eliminate` neither needs a special ordering on the basering nor a Groebner basis as input. Moreover, `eliminate` does not work in non-commutative quotients.

Remark: in a non-commutative algebra, not every subset of a set of variables generates a proper subalgebra. But if it is so, there may be cases, when no elimination (by means of Groebner bases) is possible; in such situations error messages will be reported. See also Section 7.5.19 [ncpreim_lib], page 554 for the advanced algorithm for elimination and preimage.

Example:

```
ring r=0,(e,f,h,a),Dp;
matrix d[4][4];
d[1,2]=-h; d[1,3]=2*e; d[2,3]=-2*f;
def R=nc_algebra(1,d); setring R;
// this algebra is U(sl_2), tensored with K[a] over K
option(redSB);
option(redTail);
```

```
poly  p = 4*e*f+h^2-2*h - a;
// p is a central element with parameter
ideal I = e^3, f^3, h^3-4*h, p; // take this ideal
// and intersect I with the ring K[a]
ideal J = eliminate(I,e*f*h);
// if we want substitute 'a' with a value,
// it has to be a root of this polynomial
J;
↦ J[1]=a3-32a2+192a
// now we try to eliminate h,
// that is we intersect I with the subalgebra S,
// generated by e and f.
// But S is not closed in itself, since f*e-e*f=-h !
// the next command will definitely produce an error
eliminate(I,h);
↦    ? no elimination is possible: subalgebra is not admissible
↦    ? error occurred in or before ./examples/eliminate_(plural).sing l\
   ine 21: ‘eliminate(I,h);‘
// since a commutes with e,f,h, we can eliminate it:
eliminate(I,a);
↦ _[1]=h3-4h
↦ _[2]=fh2-2fh
↦ _[3]=f3
↦ _[4]=eh2+2eh
↦ _[5]=2efh-h2-2h
↦ _[6]=e3
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.2.3 [module (plural)], page 325; Section 7.3.26 [std (plural)], page 360.

7.3.6 envelope

Syntax: `envelope` (ring_name)

Type: ring

Purpose: creates an enveloping algebra of a given algebra, that is $A^{env} = A \otimes_K A^{opp}$, where A^{opp} is the opposite algebra of A.

Remark: You have to activate the ring with the `setring` command. For the presentation, see explanation of `opposite` in Section 7.3.20 [opposite], page 353.

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
def Aenv = envelope(A);
```

```
setring Aenv;
Aenv;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering a
↦ //                  : names    H F E
↦ //                  : weights  1 1 1
↦ //        block   3 : ordering ls
↦ //                  : names    H F E
↦ //        block   4 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
↦ //    FH=HF-2F
↦ //    EH=HE+2E
↦ //    EF=FE-H
```

See Section 7.3.19 [oppose], page 352; Section 7.3.20 [opposite], page 353.

7.3.7 fetch (plural)

Syntax: `fetch ( ring_name, name )`

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: maps objects between rings. `fetch` is the identity map between rings and qrings, the i-th variable of the source ring is mapped to the i-th variable of the basering. The coefficient fields must be compatible. (See Section 7.2.2 [map (plural)], page 322 for a description of possible mappings between different ground fields).

`fetch` offers a convenient way to change variable names or orderings, or to map objects from a ring to a quotient ring of that ring or vice versa.

Note: Compared with `imap`, `fetch` uses the position of the ring variables, not their names.

Example:

```
LIB "ncalg.lib";
def Usl2 = makeUsl2(); // this algebra is U(sl_2)
setring Usl2;
option(redSB);
option(redTail);
poly  C  = 4*e*f+h^2-2*h; // the central element of Usl2
ideal I  = e^3,f^3,h^3-4*h;
ideal J  = twostd(I);
 // print a compact presentation of J:
print(matrix(ideal(J[1..5]))); // first 5 generators
↦ h3-4h,fh2-2fh,eh2+2eh,f2h-2f2,2efh-h2-2h
print(matrix(ideal(J[6..size(J)]))); // last generators
↦ e2h+2e2,f3,ef2-fh,e2f-eh-2e,e3
ideal QC = twostd(C-8);
qring Q  = QC;
ideal QJ = fetch(Usl2,J);
```

```
QJ = std(QJ);
// thus QJ is the image of I in the factor-algebra QC
print(matrix(QJ)); // print QJ compactly
↦ h3-4h,fh2-2fh,eh2+2eh,f2h-2f2,e2h+2e2,f3,e3
```

See Section 7.3.8 [imap (plural)], page 341; Section 7.2.2 [map (plural)], page 322; Section 7.2.5 [qring (plural)], page 329; Section 7.2.7 [ring (plural)], page 332.

7.3.8 imap (plural)

Syntax: `imap` (ring_name, name)

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: identity map on common subrings. `imap` is the map between rings and qrings with compatible ground fields which is the identity on variables and parameters of the same name and 0 otherwise. (See Section 7.2.2 [map (plural)], page 322 for a description of possible mappings between different ground fields). Useful for mappings from a homogenized ring to the original ring or for mappings from/to rings with/without parameters. Compared with `fetch`, `imap` uses the names of variables and parameters. **Unlike `map` and `fetch`, `imap` can map parameters to variables.**

Example:

```
LIB "ncalg.lib";
ring ABP=0,(p4,p5,a,b),dp; //  a commutative ring
def Usl3 = makeUsl(3);
def BIG  = Usl3+ABP;
setring BIG;
poly P4 = 3*x(1)*y(1)+3*x(2)*y(2)+3*x(3)*y(3);
P4 = P4 +h(1)^2+h(1)*h(2)+h(2)^2-3*h(1)-3*h(2);
// P4 is a central element of Usl3 of degree 2
poly P5 = 4*x(1)*y(1) + h(1)^2 - 2*h(1);
// P5 is a central element of the subalgebra of U(sl_3),
// generated by x(1),y(1),h(1)
ideal J = x(1),x(2),h(1)-a,h(2)-b;
// we are interested in the module U(sl_3)/J,
// which depends on parameters a,b
ideal I = p4-P4, p5-P5;
ideal K = I, J;
ideal E = eliminate(K,x(1)*x(2)*x(3)*y(1)*y(2)*y(3)*h(1)*h(2));
E; // this is the ideal of central characters in ABP
↦ E[1]=a*b+b^2-p4+p5+a+3*b
↦ E[2]=a^2-p5+2*a
↦ E[3]=b^3+p4*a-p5*a-a^2-p4*b+3*b^2
// what are the characters on nonzero a,b?
ring abP = (0,a,b),(p4,p5),dp;
ideal abE = imap(BIG, E);
option(redSB);
option(redTail);
abE = std(abE);
// here come characters (indeed, we have only one)
// that is a maximal ideal in K[p4,p5]
abE;
```

```
↦ abE[1]=p5+(-a^2-2*a)
↦ abE[2]=p4+(-a^2-a*b-3*a-b^2-3*b)
```

See Section 7.3.7 [fetch (plural)], page 340; Section 7.2.2 [map (plural)], page 322; Section 7.2.5 [qring (plural)], page 329; Section 7.2.7 [ring (plural)], page 332.

7.3.9 intersect (plural)

Syntax: `intersect` (expression_list of ideal_expression)
`intersect` (expression_list of module_expression)

Type: ideal, resp. module

Purpose: computes the intersection of ideals, resp. modules.

Example:

```
ring r=0,(x,y),dp;
def R=nc_algebra(-1,0);  //anti-commutative algebra
setring R;
module M=[x,x],[y,0];
module N=[0,y^2],[y,x];
option(redSB);
module Res;
Res=intersect(M,N);
print(Res);
↦ y2, 0,
↦ -xy,xy2
kill r,R;
//--------------------------------
LIB "ncalg.lib";
ring r=0,(x,d),dp;
def RR=Weyl(); // make r into Weyl algebra
setring RR;
ideal I = x+d^2;
ideal J = d-1;
ideal H = intersect(I,J);
H;
↦ H[1]=d4+xd2-2d3-2xd+d2+x+2d-2
↦ H[2]=xd3+x2d-xd2+d3-x2+xd-2d2-x+1
```

7.3.10 kbase (plural)

Syntax: `kbase` (ideal_expression)
`kbase` (module_expression)
`kbase` (ideal_expression`,` int_expression)
`kbase` (module_expression`,` int_expression)

Type: the same as the input type of the first argument

Purpose: with one argument: computes the vector space basis of the factor-module that equals ring (resp. free module) modulo the ideal (resp. submodule), generated by the initial terms of the given generators.
If the factor-module is not of finite dimension, -1 is returned.

If the generators form a Groebner basis, this is the same as the vector space basis of the factor-module.

when called with two arguments: computes the part of a vector space basis of the respective quotient with degree (of monomials) equal to the second argument. Here, the quotient does not need to be finite dimensional.

Note: in the non-commutative case, a ring modulo an ideal has a ring structure if and only if the ideal is two-sided.

Also, `kbase` respects module grading given by the `isHomog` attribute of input modules.

Example:

```
ring r=0,(x,y,z),dp;
matrix d[3][3];
d[1,2]=-z;  d[1,3]=2x;  d[2,3]=-2y;
def R=nc_algebra(1,d); // this algebra is U(sl_2)
setring R;
ideal i=x2,y2,z2-1;
i=std(i);
print(matrix(i));  // print a compact presentation of i
↦ z2-1,yz-y,xz+x,y2,2xy-z-1,x2
kbase(i);
↦ _[1]=z
↦ _[2]=y
↦ _[3]=x
↦ _[4]=1
vdim(i);
↦ 4
ideal j=x,z-1;
j=std(j);
kbase(j,3);
↦ _[1]=y3
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.2.3 [module (plural)], page 325; Section 7.3.30 [vdim (plural)], page 364.

7.3.11 lift (plural)

Syntax: `lift (` ideal_expression`,` subideal_expression `)`
`lift (` module_expression`,` submodule_expression `)`

Type: matrix

Purpose: computes the (left) transformation matrix which expresses the (left) generators of a submodule in terms of the (left) generators of a module. Uses different algorithms for modules which are (resp. are not) represented by Groebner bases.
More precisely, if `m` is the module, `sm` the submodule, and `T` the transformation matrix returned by lift, then `transpose(matrix(sm)) = transpose(T)*transpose(matrix(m))`.

If `m` and `sm` are ideals, `ideal(sm) = ideal(transpose(T)*transpose(matrix(m)))`.

Note: Gives a warning if `sm` is not a submodule.

Example:

```
ring r = (0,a),(e,f,h),(c,dp);
matrix D[3][3];
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
```

```
def R=nc_algebra(1,D); // this algebra is a parametric U(sl_2)
setring R;
ideal I = e,h-a; // consider this parametric ideal
I = std(I); // left Groebner basis
print(matrix(I)); // print a compact presentation of I
↦ h+(-a),e
poly Z = 4*e*f+h^2-2*h; // a central element in R
Z = Z - NF(Z,I); // a central character
ideal j = std(Z);
j;
↦ j[1]=4*ef+h2-2*h+(-a2-2a)
matrix T = lift(I,j);
print(T);
↦ h+(a+2),
↦ 4*f
ideal tj = ideal(transpose(T)*transpose(matrix(I)));
size(ideal(matrix(j)-matrix(tj))); // test for 0
↦ 0
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.3.12 [liftstd (plural)], page 344; Section 7.2.3 [module (plural)], page 325.

7.3.12 liftstd (plural)

Syntax: `liftstd` (ideal_expression, matrix_name)
`liftstd` (module_expression, matrix_name)
`liftstd` (ideal_expression, matrix_name, module_name)
`liftstd` (module_expression, matrix_name, module_name)

Type: ideal or module

Purpose: returns a left Groebner basis of an ideal or module and a left transformation matrix from the given ideal, resp. module, to the Groebner basis.
That is, if `m` is the ideal or module, `sm` is the left Groebner basis of `m`, returned by `liftstd`, and `T` is a left transformation matrix, then `sm=module(transpose(transpose(T)*transpose(matrix(m))))`.
If `m` is an ideal, `sm=ideal(transpose(T)*transpose(matrix(m)))`.
In an optional third argument the left syzygy module will be returned.

Example:

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A;  // this algebra is U(sl_2)
ideal i = e2,f;
option(redSB);
option(redTail);
matrix T;
ideal j = liftstd(i,T);
// the Groebner basis in a compact form:
print(matrix(j));
↦ f,2h2+2h,2eh+2e,e2
print(T);  // the transformation matrix
↦ 0,f2,         -f,1,
↦ 1,-e2f+4eh+8e,e2,0
```

```
ideal tj = ideal(transpose(T)*transpose(matrix(i)));
size(ideal(matrix(j)-matrix(tj))); // test for 0
↦ 0
module S; ideal k = liftstd(i,T,S); // the third argument
S = std(S); print(S); // the syzygy module
↦ -ef-2h+6,-f3,
↦ e3,      e2f2-6efh-6ef+6h2+18h+12
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.2.7 [ring (plural)], page 332; Section 7.3.26 [std (plural)], page 360.

7.3.13 minres (plural)

Syntax: `minres` (list_expression)

Type: list

Syntax: `minres` (resolution_expression)

Type: resolution

Purpose: minimizes a free resolution of an ideal or module given by the list_expression, resp. resolution_expression.

Example:

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; // this algebra is U(sl_2)
ideal i=e,f,h;
i=std(i);
resolution F=nres(i,0); F;
↦  1      3      3      1
↦ A <--   A <--   A <--   A
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
list lF = F; lF;
↦ [1]:
↦    _[1]=h
↦    _[2]=f
↦    _[3]=e
↦ [2]:
↦    _[1]=f*gen(1)-h*gen(2)-2*gen(2)
↦    _[2]=e*gen(1)-h*gen(3)+2*gen(3)
↦    _[3]=e*gen(2)-f*gen(3)-gen(1)
↦ [3]:
↦    _[1]=e*gen(1)-f*gen(2)+h*gen(3)
print(betti(lF), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     3     1
↦ ------------------------------
↦ total:     1     0     3     1
↦
resolution MF=minres(F);  MF;
```

```
↦  1      2      2      1
↦ A <--  A <--  A <--  A
↦
↦ 0      1      2      3
↦
list lMF = F; lMF;
↦ [1]:
↦    _[1]=f
↦    _[2]=e
↦ [2]:
↦    _[1]=-ef*gen(1)+f2*gen(2)+2h*gen(1)+2*gen(1)
↦    _[2]=-e2*gen(1)+ef*gen(2)+h*gen(2)-2*gen(2)
↦ [3]:
↦    _[1]=e*gen(1)-f*gen(2)
print(betti(lMF), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     -     2     1
↦ ------------------------------
↦ total:     1     0     2     1
↦
```

See Section 7.3.15 [mres (plural)], page 347; Section 7.3.18 [nres (plural)], page 350.

7.3.14 modulo (plural)

Syntax: `modulo` (ideal_expression, ideal_expression)
`modulo` (module_expression, module_expression)

Type: module

Purpose: `modulo(h1,h2)` represents $h_1/(h_1 \cap h_2) \cong (h_1 + h_2)/h_2$, where h_1 and h_2 are considered as submodules of the same free module R^l (l=1 for ideals).
Let H_1 (resp. H_2) be the matrix of size $l \times k$ (resp. $l \times m$), having the generators of h_1 (resp. h_2) as columns.
Then $h_1/(h_1 \cap h_2) \cong R^k/ker(\overline{H_1})$, where $\overline{H_1} : R^k \to R^l/Im(H_2) = R^l/h_2$ is the induced map given by H_1.
`modulo(h1,h2)` returns generators of the kernel of this induced map.

Note: `If, for at least one of`
h_1 or h_2, the attribute `isHomog` is st, then `modulo(h1,h2)` also sets this attribute (if the weights are compatible).

Example:

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; // this algebra is U(sl_2)
option(redSB);
option(redTail);
ideal H2 = e2,f2,h2-1;
H2 = twostd(H2);
print(matrix(H2)); // print H2 in a compact form
↦ h2-1,fh-f,eh+e,f2,2ef-h-1,e2
ideal H1 = std(e);
ideal T = modulo(H1,H2);
T = NF(std(H2+T),H2);
T = std(T);
T;
↦ T[1]=h-1
↦ T[2]=e
```

See also Section 7.3.28 [syz (plural)], page 362.

7.3.15 mres (plural)

Syntax: `mres` (ideal_expression, int_expression)
`mres` (module_expression, int_expression)

Type: resolution

Purpose: computes a minimal free resolution of an ideal or module M with the Groebner basis method. More precisely, let A=`matrix`(M), then `mres` computes a free resolution of $coker(A) = F_0/M$

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0,$$

where the columns of the matrix A_1 are a (possibly) minimal set of generators of M . If the int expression k is not zero, then the computation stops after k steps and returns a resolution consisting of modules $M_i =$ `module`(A_i), $i = 1 \ldots k$.
`mres(M,0)` returns a resolution consisting of at most n+2 modules, where n is the number of variables of the basering. Let `list L=mres(M,0);` then `L[1]` consists of a minimal set of generators `M`, `L[2]` consists of a minimal set of generators for the first syzygy module of `L[1]`, etc., until `L[p+1]`, such that `L[i]` $\neq 0$ for $i \leq p$, but `L[p+1]` (the first syzygy module of `L[p]`) is 0 (if the basering is not a qring).

Note: Accessing single elements of a resolution may require that some partial computations have to be finished and may therefore take some time. Hence, assigning right away to a list is the recommended way to do it.

Example:

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; // this algebra is U(sl_2)
option(redSB);
option(redTail);
ideal i = e,f,h;
i = std(i);
resolution M=mres(i,0);
M;
↦  1      2      2      1
↦ A <--  A <--  A <--  A
↦
↦ 0      1      2      3
↦
list l = M; l;
↦ [1]:
↦    _[1]=f
↦    _[2]=e
↦ [2]:
↦    _[1]=ef*gen(1)-f2*gen(2)-2h*gen(1)-2*gen(1)
↦    _[2]=e2*gen(1)-ef*gen(2)-h*gen(2)+2*gen(2)
↦ [3]:
↦    _[1]=e*gen(1)-f*gen(2)
// see the exactness at this point
size(ideal(transpose(matrix(l[2]))*transpose(matrix(l[1]))));
↦ 0
print(matrix(M[3]));
↦ e,
↦ -f
// see the exactness at this point
size(ideal(transpose(matrix(l[3]))*transpose(matrix(l[2]))));
↦ 0
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.3.13 [minres (plural)], page 345; Section 7.2.3 [module (plural)], page 325; Section 7.3.18 [nres (plural)], page 350.

7.3.16 nc_algebra

Syntax:

`nc_algebra(` matrix_expression C, matrix_expression D)
`nc_algebra(` number_expression n, matrix_expression D)
`nc_algebra(` matrix_expression C, poly_expression p)
`nc_algebra(` number_expression n, poly_expression p)

Type: ring

Purpose: Executed in the basering `r`, say, in k variables $x_1, \ldots, x_k$, `nc_algebra` creates and returns the non-commutative extension of `r` subject to relations $\{x_j x_i = c_{ij} \cdot x_i x_j + d_{ij}, 1 \leq i < j \leq k\}$, where c_{ij} and d_{ij} must be put into two strictly upper triangular matrices `C` with entries c_{ij} from the ground field of `r` and `D` with (commutative) polynomial entries d_{ij} from `r`. See all the details in Section 7.4.1 [G-algebras], page 365.
If $\forall i < j$, $c_{ij} = n$, one can input the number `n` instead of matrix `C`.
If $\forall i < j$, $d_{ij} = p$, one can input the polynomial `p` instead of matrix `D`.

Note: `The returned ring should be activated afterwards, using the command setring.`

Note: `The coefficients must be a field (see Section 7.4.1 [G-algebras], page 365).`

Remark: At present, Plural does not check the non-degeneracy conditions (see Section 7.4.1 [G-algebras], page 365) while setting an algebra.

Example:

```
LIB "nctools.lib";
// ------- first example: C, D are matrices --------
ring r1 = (0,Q),(x,y,z),Dp;
minpoly = rootofUnity(6);
matrix C[3][3];
matrix D[3][3];
C[1,2]=Q2;   C[1,3]=1/Q2;  C[2,3]=Q2;
D[1,2]=-Q*z; D[1,3]=1/Q*y; D[2,3]=-Q*x;
def S=nc_algebra(C,D);
// this algebra is a quantum deformation U'_q(so_3),
// where Q is a 6th root of unity
setring S;S;
```

```
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering Dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
kill r1,S;
// ----- second example: number n=1, D is a matrix
ring r2=0,(Xa,Xb,Xc,Ya,Yb,Yc,Ha,Hb),dp;
matrix d[8][8];
d[1,2]=-Xc; d[1,4]=-Ha; d[1,6]=Yb;  d[1,7]=2*Xa;
d[1,8]=-Xa; d[2,5]=-Hb; d[2,6]=-Ya; d[2,7]=-Xb;
d[2,8]=2*Xb; d[3,4]=Xb; d[3,5]=-Xa; d[3,6]=-Ha-Hb;
d[3,7]=Xc;   d[3,8]=Xc; d[4,5]=Yc; d[4,7]=-2*Ya;
d[4,8]=Ya;  d[5,7]=Yb; d[5,8]=-2*Yb;
d[6,7]=-Yc; d[6,8]=-Yc;
def S=nc_algebra(1,d);  // this algebra is U(sl_3)
setring S;S;
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering dp
↦ //                  : names    Xa Xb Xc Ya Yb Yc Ha Hb
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    XbXa=Xa*Xb-Xc
↦ //    YaXa=Xa*Ya-Ha
↦ //    YcXa=Xa*Yc+Yb
↦ //    HaXa=Xa*Ha+2*Xa
↦ //    HbXa=Xa*Hb-Xa
↦ //    YbXb=Xb*Yb-Hb
↦ //    YcXb=Xb*Yc-Ya
↦ //    HaXb=Xb*Ha-Xb
↦ //    HbXb=Xb*Hb+2*Xb
↦ //    YaXc=Xc*Ya+Xb
↦ //    YbXc=Xc*Yb-Xa
↦ //    YcXc=Xc*Yc-Ha-Hb
↦ //    HaXc=Xc*Ha+Xc
↦ //    HbXc=Xc*Hb+Xc
↦ //    YbYa=Ya*Yb+Yc
↦ //    HaYa=Ya*Ha-2*Ya
↦ //    HbYa=Ya*Hb+Ya
↦ //    HaYb=Yb*Ha+Yb
↦ //    HbYb=Yb*Hb-2*Yb
↦ //    HaYc=Yc*Ha-Yc
↦ //    HbYc=Yc*Hb-Yc
kill r2,S;
// ---- third example: C is a matrix, p=0 is a poly
ring r3=0,(a,b,c,d),lp;
matrix c[4][4];
c[1,2]=1; c[1,3]=3; c[1,4]=-2;
```

```
c[2,3]=-1; c[2,4]=-3; c[3,4]=1;
def S=nc_algebra(c,0); // it is a quasi--commutative algebra
setring S;S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    a b c d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ca=3ac
↦ //    da=-2ad
↦ //    cb=-bc
↦ //    db=-3bd
kill r3,S;
// -- fourth example : number n = -1, poly p = 3w
ring r4=0,(u,v,w),dp;
def S=nc_algebra(-1,3w);
setring S;S;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    u v w
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    vu=-uv+3w
↦ //    wu=-uw+3w
↦ //    wv=-vw+3w
kill r4,S;
```

See also Section 7.5.10 [ncalg_lib], page 464; Section 7.5.20 [nctools_lib], page 562; Section 7.5.24 [qmatrix_lib], page 609.

7.3.17 ncalgebra

Syntax:

`ncalgebra(` matrix_expression C, matrix_expression D)
`ncalgebra(` number_expression n, matrix_expression D)
`ncalgebra(` matrix_expression C, poly_expression p)
`ncalgebra(` number_expression n, poly_expression p)

Type: none

Purpose: Works like Section 7.3.16 [nc_algebra], page 348 but changes the basering.

Remark: This function is **deprecated** and should be substituted by `nc_algebra`, since it violates the general SINGULAR policy: only Section 4.20 [ring], page 127 and Section 5.1.141 [setring], page 260 can change the basering. More concretely, replace by `def A = nc_algebra(C, D); setring A;` which will additionally introduce a new ring. Afterwards, some objects may have to be mapped into the new ring.

See also Section 7.3.16 [nc_algebra], page 348; Section 7.5.20 [nctools_lib], page 562.

7.3.18 nres (plural)

Syntax: `nres` (ideal_expression, int_expression)
`nres` (module_expression, int_expression)

Type: resolution

Purpose: computes a free resolution of an ideal or module which is minimized from the second module on (by the Groebner basis method).

Note: `Assigning a resolution to a list is the best choice of usage. The resolution may be minimized by using the` command `minres`. Use the command `betti` to compute Betti numbers.

Example:

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; // this algebra is U(sl_2)
option(redSB);
option(redTail);
ideal i = e,f,h;
i = std(i);
resolution F=nres(i,0); F;
↦  1      3      3      1
↦ A <--  A <--  A <--  A
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
list l = F; l;
↦ [1]:
↦    _[1]=h
↦    _[2]=f
↦    _[3]=e
↦ [2]:
↦    _[1]=f*gen(1)-h*gen(2)-2*gen(2)
↦    _[2]=e*gen(1)-h*gen(3)+2*gen(3)
↦    _[3]=e*gen(2)-f*gen(3)-gen(1)
↦ [3]:
↦    _[1]=e*gen(1)-f*gen(2)+h*gen(3)
// see the exactness at this point:
size(ideal(transpose(matrix(l[2]))*transpose(matrix(l[1]))));
↦ 0
// see the exactness at this point:
size(ideal(transpose(matrix(l[3]))*transpose(matrix(l[2]))));
↦ 0
print(betti(l), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     3     1
↦ ------------------------------
↦ total:     1     0     3     1
↦
print(betti(minres(l)), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦      1:      -      -      2      1
↦ ------------------------------
↦ total:      1      0      2      1
↦
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.3.13 [minres (plural)], page 345; Section 7.2.3 [module (plural)], page 325; Section 7.3.15 [mres (plural)], page 347.

7.3.19 oppose

Syntax: oppose (ring_name, name)

Type: poly, vector, ideal, module or matrix (the same type as the second argument)

Purpose: for a given object in the given ring, creates its opposite object in the opposite (Section 7.3.20 [opposite], page 353) ring (the last one is assumed to be the current ring).

Remark: for any object O, $(O^{opp})^{opp} = O$.

```
LIB "ncalg.lib";
def R = makeUsl2();
setring R;
matrix m[3][4];
poly   p  = (h^2-1)*f*e;
vector v  = [1,e*h,0,p];
ideal  i  = h*e, f^2*e,h*f*e;
m         = e,f,h,1,0,h^2, p,0,0,1,e^2,e*f*h+1;
module mm = module(m);
def b     = opposite(R);
setring b; b;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering a
↦ //                  : names    H F E
↦ //                  : weights  1 1 1
↦ //        block   2 : ordering ls
↦ //                  : names    H F E
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    FH=HF-2F
↦ //    EH=HE+2E
↦ //    EF=FE-H
// we will oppose these objects: p,v,i,m,mm
poly P    = oppose(R,p);
vector V  = oppose(R,v);
ideal  I  = oppose(R,i);
matrix M  = oppose(R,m);
module MM = oppose(R,mm);
def c = opposite(b);
setring c;   // now let's check the correctness:
// print compact presentations of objects
print(oppose(b,P)-imap(R,p));
↦ 0
print(oppose(b,V)-imap(R,v));
↦ [0]
print(matrix(oppose(b,I))-imap(R,i));
```

```
↦ 0,0,0
print(matrix(oppose(b,M))-imap(R,m));
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
print(matrix(oppose(b,MM))-imap(R,mm));
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
```

See Section 7.3.6 [envelope], page 339; Section 7.3.20 [opposite], page 353.

7.3.20 opposite

Syntax: `opposite` (ring_name)

Type: ring

Purpose: creates an opposite algebra of a given algebra.

Note: activate the ring with the `setring` command.
An opposite algebra of a given algebra (A ,#) is an algebra (A ,*) with the same vector space but with the opposite multiplication, i.e.
$\forall\ f, g \in A^{opp}$, a new multiplication $*$ on A^{opp} is defined to be $f * g := g \sharp f$.

This is an identity functor on commutative algebras.

Remark: Starting from the variables x_1,...,x_N and the ordering `<` of the given algebra, an opposite algebra will have variables X_N,...,X_1 (where the case and the position are reverted). Moreover, it is equipped with an opposed ordering `<_opp` (it is given by the matrix, obtained from the matrix ordering of `<` with the reverse order of columns).
Currently not implemented for non-global orderings.

```
LIB "ncalg.lib";
def B = makeQso3(3);
// this algebra is a quantum deformation of U(so_3),
// where the quantum parameter is a 6th root of unity
setring B; B;
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
def Bopp = opposite(B);
setring Bopp;
Bopp;
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering a
↦ //                  : names    Z Y X
↦ //                  : weights  1 1 1
↦ //        block   2 : ordering ls
↦ //                  : names    Z Y X
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    YZ=(Q-1)*ZY+(-Q)*X
↦ //    XZ=(-Q)*ZX+(-Q+1)*Y
↦ //    XY=(Q-1)*YX+(-Q)*Z
def Bcheck = opposite(Bopp);
setring Bcheck; Bcheck;  // check that (B-opp)-opp = B
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering wp
↦ //                  : names    x y z
↦ //                  : weights  1 1 1
↦ //        block   2 : ordering C
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
```

See Section B.2.6 [Matrix orderings], page 770; Section 7.3.6 [envelope], page 339; Section 7.3.19 [oppose], page 352.

7.3.21 preimage (plural)

Syntax: `preimage` (ring_name, map_name, ideal_name)
`preimage` (ring_name, ideal_expression, ideal_name)

Type: ideal

Purpose: returns the preimage of an ideal under a given map. The second argument has to be a map from the basering to the given ring (or an ideal defining such a map), and the ideal has to be an ideal in the given ring.

Note: To compute the kernel of a map, the preimage of zero has to be determined. Hence there is no special command for computing the kernel of a map in Plural.

Remark: In the non-commutative case, the command `preimage` is implemented only for maps A -> B , where A is a commutative ring. See Section 7.5.19 [ncpreim_lib], page 554 for the most general available implementation.

Example:

```
LIB "ncalg.lib";
ring   R = 0,a,dp;
def Usl2 = makeUsl2();
setring Usl2;
poly  C = 4*e*f+h^2-2*h;
// C is a central element of U(sl2)
ideal I = e^3, f^3, h^3-4*h;
ideal Z = 0;  // zero
ideal J = twostd(I); // two-sided GB
ideal K = std(I);    // left GB
map Phi = R,C;  // phi maps a (in R) to C (in U(sl2))
setring R;
ideal PreJ = preimage(Usl2,Phi,J);
```

```
// the central character of J
PreJ;
↦ PreJ[1]=a2-8a
factorize(PreJ[1],1);
↦ _[1]=a
↦ _[2]=a-8
// hence, there are two simple characters for J
ideal PreK = preimage(Usl2,Phi,K);
// the central character of K
PreK;
↦ PreK[1]=a3-32a2+192a
factorize(PreK[1],1);
↦ _[1]=a
↦ _[2]=a-24
↦ _[3]=a-8
// hence, there are three simple characters for K
preimage(Usl2, Phi, Z);  // kernel of phi
↦ _[1]=0
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.2.2 [map (plural)], page 322; Section 7.2.7 [ring (plural)], page 332.

7.3.22 quotient (plural)

Syntax: `quotient` (ideal_expression`,` ideal_expression)
`quotient` (module_expression`,` module_expression)

Type: ideal

Syntax: `quotient` (module_expression`,` ideal_expression)

Type: module

Purpose: computes the ideal quotient, resp. module quotient. Let `R` be the basering, `I,J` ideals and `M, N` submodules in `R`^{n}. Then

`quotient(I,J)`$= \{a \in R \mid aJ \subset I\}$,

`quotient(M,J)`$= \{b \in R^n \mid bJ \subset M\}$.

Note: It can only be used for two-sided ideals (bimodules) in the second argument, otherwise the result may have no meaning.

Example:

```
//------ a very simple example ------------
ring r=(0,q),(x,y),Dp;
def R=nc_algebra(q,0); // this algebra is a quantum plane
setring R;
option(returnSB);
poly f1  = x^3+2*x*y^2+2*x^2*y;
poly f2  = y;
poly f3 = x^2;
poly f4 = x+y;
ideal i = f1,f2;
ideal I = twostd(i);
ideal j = f3,f4;
ideal J = twostd(j);
quotient(I,J);
↦ _[1]=y
↦ _[2]=x2
module M = x*freemodule(3), y*freemodule(2);
quotient(M, ideal(x,y));
↦ _[1]=gen(1)
↦ _[2]=gen(2)
↦ _[3]=x*gen(3)
kill r,R;
//------- a bit more involved example
LIB "ncalg.lib";
def Usl2 = makeUsl2();
// this algebra is U(sl_2)
setring Usl2;
ideal i = e3,f3,h3-4*h;
ideal I = std(i);
poly  C = 4*e*f+h^2-2*h;
ideal H = twostd(C-8);
option(returnSB);
ideal Q = quotient(I,H);
// print a compact presentation of Q:
print(matrix(Q));
↦ h,f3,ef2-4f,e2f-6e,e3
```

See Section 7.2.1 [ideal (plural)], page 318; Section 7.2.3 [module (plural)], page 325.

7.3.23 reduce (plural)

Syntax:

`reduce` (poly_expression, ideal_expression)
`reduce` (poly_expression, ideal_expression, int_expression)
`reduce` (vector_expression, ideal_expression)
`reduce` (vector_expression, ideal_expression, int_expression)
`reduce` (vector_expression, module_expression)
`reduce` (vector_expression, module_expression, int_expression)
`reduce` (ideal_expression, ideal_expression)
`reduce` (ideal_expression, ideal_expression, int_expression)
`reduce` (module_expression, ideal_expression)
`reduce` (module_expression, ideal_expression, int_expression)
`reduce` (module_expression, module_expression)
`reduce` (module_expression, module_expression, int_expression)

Type: the type of the first argument

Purpose: reduces a polynomial, vector, ideal or module to its **left** normal form with respect to an ideal or module represented by a left Groebner basis, if the second argument is a left Groebner basis.
returns 0 if and only if the polynomial (resp. vector, ideal, module) is an element (resp. subideal, submodule) of the ideal (resp. module).
Otherwise, the result may have no meaning.
The third (optional) argument 1 of type int forces a reduction which considers only the leading term and does no tail reduction.

Note: The commands reduce and NF are synonymous.

Example:

```
ring r=(0,a),(e,f,h),Dp;
matrix d[3][3];
d[1,2]=-h;  d[1,3]=2e;  d[2,3]=-2f;
def R=nc_algebra(1,d);
setring R;
// this algebra is U(sl_2) over Q(a)
ideal I = e2, f2, h2-1;
I = std(I);
// print a compact presentation of I
print(matrix(I));
↦ h2-1,fh-f,f2,eh+e,2*ef-h2-h,e2
ideal J = e, h-a;
J = std(J);
// print a compact presentation of J
print(matrix(J));
↦ h+(-a),e
poly z=4*e*f+h^2-2*h;
// z is the central element of U(sl_2)
reduce(z,I);  // the central character of I:
↦ 3
reduce(z,J); // the central character of J:
↦ (a2+2a)
poly nz = z - NF(z,J); // nz will belong to J
reduce(nz,J);
↦ 0
reduce(I,J);
↦ _[1]=(a2-1)
↦ _[2]=(a-1)*f
↦ _[3]=f2
↦ _[4]=0
↦ _[5]=(-a2+a)
↦ _[6]=0
```

See also Section 7.2.1 [ideal (plural)], page 318; Section 7.2.3 [module (plural)], page 325; Section 7.3.26 [std (plural)], page 360.

7.3.24 ringlist (plural)

Syntax: ringlist (ring_expression)
ringlist (qring_expression)

Type: list

Purpose: decomposes a ring/qring into a list of 6 (or 4 in the commutative case) components. The first 4 components are common both for the commutative and for the non-commutative cases, the 5th and the 6th appear only in the non-commutative case.

5. upper triangle square matrix with nonzero upper triangle, containing structural coefficients of a G-algebra (this corresponds to the matrix C from the definition of Section 7.4.1 [G-algebras], page 365)
6. square matrix, containing structural polynomials of a G-algebra (this corresponds to the matrix D from the definition of Section 7.4.1 [G-algebras], page 365)

Note: After modifying a list acquired with `ringlist`, one can construct a corresponding ring with `ring(list)`.

Example:

```
// consider the quantized Weyl algebra
ring r = (0,q),(x,d),Dp;
def RS=nc_algebra(q,1);
setring RS; RS;
↦ // coefficients: QQ(q)
↦ // number of vars : 2
↦ //        block   1 : ordering Dp
↦ //                  : names    x d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    dx=(q)*xd+1
list l = ringlist(RS);
l;
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       [1]:
↦          q
↦    [3]:
↦       [1]:
↦          [1]:
↦             lp
↦          [2]:
↦             1
↦    [4]:
↦       _[1]=0
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
↦       d
↦ [3]:
↦    [1]:
↦       [1]:
↦          Dp
↦       [2]:
↦          1,1
↦    [2]:
↦       [1]:
↦          C
↦       [2]:
↦          0
↦ [4]:
↦    _[1]=0
↦ [5]:
↦    _[1,1]=0
↦    _[1,2]=(q)
↦    _[2,1]=0
↦    _[2,2]=0
↦ [6]:
↦    _[1,1]=0
↦    _[1,2]=1
↦    _[2,1]=0
↦    _[2,2]=0
// now, change the relation d*x = q*x*d +1
// into the relation d*x=(q2+1)*x*d + q*d + 1
matrix S = l[5]; // matrix of coefficients
S[1,2] = q^2+1;
l[5] = S;
matrix T = l[6]; // matrix of polynomials
T[1,2] = q*d+1;
l[6] = T;
def rr = ring(l);
setring rr; rr;
↦ // coefficients: QQ(q)
↦ // number of vars : 2
↦ //        block   1 : ordering Dp
↦ //                  : names    x d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    dx=(q2+1)*xd+(q)*d+1
```

See also Section 7.2.7 [ring (plural)], page 332; Section 5.1.137 [ringlist], page 255.

7.3.25 slimgb (plural)

Syntax: slimgb (ideal_expression)
slimgb (module_expression)

Type: same type as argument

Purpose: returns a left Groebner basis of a left ideal or module with respect to the global monomial ordering of the basering.

Note: The commutative algorithm is described in the diploma thesis of Michael Brickenstein "Neue Varianten zur Berechnung von Groebnerbasen", written 2004 under supervision of G.-M. Greuel in Kaiserslautern.

It is designed to keep polynomials or vectors slim (short with small coefficients). Currently best results are examples over function fields (parameters).

The current implementation may not be optimal for weighted degree orderings.

The program only supports the options prot, which will give protocol output and redSB for returning a reduced Groebner basis. The protocol messages of slimgb mean the following:
M[n,m] means a parallel reduction of n elements with m non-zero output elements,
b notices an exchange trick described in the thesis and
e adds a reductor with non-minimal leading term.

slimgb works for grade commutative algebras but not for general GR-algebras. Please use qslimgb instead.

For a detailed commutative example see Section A.2.3 [slim Groebner bases], page 714.

Example:

```
LIB "nctools.lib";
LIB "ncalg.lib";
def U = makeUsl(2); setring U;
// U is the U(sl_2) algebra
ideal I = e^3, f^3, h^3-4*h;
option(redSB);
ideal J = slimgb(I);
J;
↦ J[1]=h3-4h
↦ J[2]=fh2-2fh
↦ J[3]=eh2+2eh
↦ J[4]=2efh-h2-2h
↦ J[5]=f3
↦ J[6]=e3
// compare slimgb with std:
ideal K = std(I);
print(matrix(NF(K,J)));
↦ 0,0,0,0,0,0
print(matrix(NF(J,K)));
↦ 0,0,0,0,0,0
// hence both Groebner bases are equal.
// Another example for exterior algebras
ring r;
def E = Exterior(); setring E; E;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=-xy
↦ //    zx=-xz
↦ //    zy=-yz
↦ // quotient ring from ideal
↦ _[1]=z2
↦ _[2]=y2
↦ _[3]=x2
slimgb(xy+z);
↦ _[1]=yz
↦ _[2]=xz
↦ _[3]=xy+z
```

See Section 5.1.111 [option], page 234; Section 7.3.26 [std (plural)], page 360.

7.3.26 std (plural)

Syntax: `std (` ideal_expression`)`
`std (` module_expression`)`
`std (` ideal_expression`,` poly_expression `)`
`std (` module_expression`,` vector_expression `)`

Type: ideal or module

Purpose: returns a left Groebner basis (see Section 7.4.2 [Groebner bases in G-algebras], page 366 for a definition) of an ideal or module with respect to the monomial ordering of the basering.

Use an optional second argument of type poly, resp. vector, to construct the Groebner basis from an already computed one (given as the first argument) and one additional generator (the second argument).

Note: To view the progress of long running computations, use `option(prot)`. (see Section 5.1.111 [option], page 234(prot)).

Example:

```
LIB "ncalg.lib";
option(prot);
def R = makeUsl2();
// this algebra is U(sl_2)
setring R;
ideal I = e2, f2, h2-1;
I=std(I);
↦ 2(2)s
↦ s
↦ s
↦ 3s
↦ (3)2(2)s
↦ s
↦ (4)(3)(2)3s
↦ 2(4)(3)(2)32product criterion:6 chain criterion:3
I;
↦ I[1]=h2-1
↦ I[2]=fh-f
↦ I[3]=eh+e
↦ I[4]=f2
↦ I[5]=2ef-h-1
↦ I[6]=e2
kill R;
//-------------------------------------------
def RQ = makeQso3(3);
// this algebra is U'_q(so_3),
// where Q is a 6th root of unity
setring RQ;
RQ;
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
ideal J=x2, y2, z2;
J=std(J);
↦ 2(2)s
↦ s
↦ s
↦ 3s
↦ (4)s
↦ 2(3)s
↦ (5)s
↦ (6)s
↦ 1(8)s
↦ (7)(5)s
↦ (3)(2)product criterion:0 chain criterion:17
J;
↦ J[1]=z
↦ J[2]=y
↦ J[3]=x
```

See also Section 7.2.1 [ideal (plural)], page 318; Section 7.2.7 [ring (plural)], page 332.

7.3.27 subst (plural)

Syntax: `subst` (poly_expression, ring_variable, poly_expression)
`subst` (vector _expression, ring_variable, poly_expression)
`subst` (ideal_expression, ring_variable, poly_expression)
`subst` (module _expression, ring_variable, poly_expression)

Type: poly, vector, ideal or module (corresponding to the first argument)

Purpose: substitutes a ring variable by a polynomial.

Example:

```
LIB "ncalg.lib";
def R = makeUsl2();
// this algebra is U(sl_2)
setring R;
poly C  = e*f*h;
poly C1 = subst(C,e,h^3);
C1;
↦ fh4-6fh3+12fh2-8fh
poly C2 = subst(C,f,e+f);
C2;
↦ e2h+efh
```

See also Section 7.2.2 [map (plural)], page 322.

7.3.28 syz (plural)

Syntax: `syz` (ideal_expression)
`syz` (module_expression)

Type: module

Purpose: computes the first syzygy (i.e., the module of relations of the given generators) of the ideal, resp. module.

Note: if `S` is a matrix of a left syzygy module of left submodule given by matrix `M`, then `transpose(S)*transpose(M) = 0`.

Example:

```
LIB "ncalg.lib";
def R = makeQso3(3); setring R;
// we wish to have completely reduced bases:
option(redSB);  option(redTail);
ideal tst;
ideal J = x3+x,x*y*z;
print(syz(J));
↦ -yz,
↦ x2+1
ideal K = x+y+z,y+z,z;
module S = syz(K);
print(S);
↦ (Q-1),        (-Q+1)*z,   (-Q)*y,
↦ (Q)*z+(-Q+1),(Q-1)*z+(Q),-x+(Q)*y,
↦ y+(-Q)*z,    x+(-Q),      x+(-Q+1)
tst = ideal(transpose(S)*transpose(K));
// check the property of a syzygy module (tst==0):
size(tst);
↦ 0
// Now compute the Groebner basis of K ...
K = std(K);
// ... print a matrix presentation of K ...
print(matrix(K));
↦ z,y,x
S = syz(K); // ... and its syzygy module
print(S);
↦ y,    x,       (Q-1),
↦ (Q)*z,(Q),     x,
↦ (Q-1),(-Q+1)*z,(Q)*y
tst = ideal(transpose(S)*transpose(K));
// check the property of a syzygy module (tst==0):
size(tst);
↦ 0
// Note the "commutative" (not transposed) syzygy property does not hold
size(ideal(matrix(K)*matrix(S)));
↦ 3
```

See also Section 7.2.1 [ideal (plural)], page 318; Section 7.3.13 [minres (plural)], page 345; Section 7.2.3 [module (plural)], page 325; Section 7.3.15 [mres (plural)], page 347; Section 7.3.18 [nres (plural)], page 350.

7.3.29 twostd (plural)

Syntax: `twostd(` ideal_expression`);`

Type: ideal

Purpose: returns a two-sided Groebner basis of an input

Note: `Treating the input as a set of`
two-sided generators of a two-sided ideal `T`, two-sided Groebner basis is a left (and a right) Groebner basis of `T`. (see Section 5.1.151 [std], page 271).

Remark: There are algebras with no two-sided ideals except 0 and the whole algebra (like Weyl algebras).

Example:

```
LIB "ncalg.lib";
def U = makeUsl2(); // this algebra is U(sl_2)
setring U;
ideal i= e^3, f^3, h^3 - 4*h;
option(redSB);
option(redTail);
ideal I = std(i);
print(matrix(I)); // print a compact presentation of I
↦ h3-4h,fh2-2fh,eh2+2eh,2efh-h2-2h,f3,e3
ideal J = twostd(i);
// print a compact presentation of J:
print(matrix(ideal(J[1..6]))); // first 6 gen’s
↦ h3-4h,fh2-2fh,eh2+2eh,f2h-2f2,2efh-h2-2h,e2h+2e2
print(matrix(ideal(J[7..size(J)]))); // the rest of gen’s
↦ f3,ef2-fh,e2f-eh-2e,e3
// compute the set of elements present in J but not in I
ideal K = NF(J,I);
K = K+0; // simplify K
print(matrix(K));
↦ f2h-2f2,e2h+2e2,ef2-fh,e2f-eh-2e
```

7.3.30 vdim (plural)

Syntax: `vdim` (ideal_expression)
`vdim` (module_expression)

Type: int

Purpose: computes the vector space dimension of the factor-module that equals ring (resp. free module) modulo the ideal (resp. submodule), generated by the leading terms of the given generators.
If the factor-module is not of finite dimension, -1 is returned.

If the generators form a left Groebner basis, this is the same as the vector space dimension of the left factor module.

Note: In the non-commutative case, a ring modulo an ideal has a ring structure if and only if the ideal is two-sided.

Example:

```
ring R=0,(x,y,z),dp;
matrix d[3][3];
d[1,2]=-z;  d[1,3]=2x;  d[2,3]=-2y;
def RS=nc_algebra(1,d); //U(sl_2)
setring RS;
option(redSB); option(redTail);
ideal I=x3,y3,z3-z;
I=std(I);
I;
↦ I[1]=z3-z
↦ I[2]=y3
↦ I[3]=x3
↦ I[4]=y2z2-y2z
↦ I[5]=x2z2+x2z
↦ I[6]=x2y2z-2xyz2-2xyz+2z2+2z
vdim(I);
↦ 21
```

See also Section 7.2.1 [ideal (plural)], page 318; Section 7.3.10 [kbase (plural)], page 342; Section 7.3.26 [std (plural)], page 360.

7.4 Mathematical background (plural)

This section introduces some of the mathematical notions and definitions used throughout the Plural manual. For details, please, refer to appropriate articles or text books (see Section 7.4.4 [References (plural)], page 369). A detailed discussion of the subjects in this section can be found in the doctoral thesis [LV] of V. Levandovskyy (see Section 7.4.4 [References (plural)], page 369).

All algebras are assumed to be associative K -algebras for some field K .

7.4.1 G-algebras

Definition (PBW basis)

Let K be a field, and let a K-algebra A be generated by variables $x_1,\ldots,x_n$ subject to some relations. We call A an algebra with **PBW basis** (Poincaré-Birkhoff-Witt basis), if a K-basis of A is $\mathrm{Mon}(x_1,\ldots,x_n)=\{x_1^{a_1}x_2^{a_2}\ldots x_n^{a_n}\mid a_i\in N\cup\{0\}\}$, where a power-product $x_1^{a_1}x_2^{a_2}\ldots x_n^{a_n}$ (in this particular order) is called **a monomial**. For example, x_1x_2 is a monomial, while x_2x_1 is, in general, not a monomial.

Definition (G-algebra)

Let K be a field, and let a K-algebra A be given in terms of generators subject to the following relations:

$A=K\langle x_1,\ldots,x_n\mid\{x_jx_i=c_{ij}\cdot x_ix_j+d_{ij}\},1\leq i<j\leq n\rangle$, where $c_{ij}\in K^*,d_{ij}\in K[x_1,\ldots,x_n]$.

A is called **a G–algebra, if the following conditions hold:**

- there is a monomial well-ordering $<$ on $K[x_1,x_2,\ldots,x_n]$ such that $\forall i<j\ \ \mathrm{LM}(d_{ij})<x_ix_j$,
- **non-degeneracy conditions**: $\forall\ 1\leq i<j<k\leq n\quad:\ \mathcal{NDC}_{ijk}=0$, where

$$\mathcal{NDC}_{ijk}=c_{ik}c_{jk}\cdot d_{ij}x_k-x_kd_{ij}+c_{jk}\cdot x_jd_{ik}-c_{ij}\cdot d_{ik}x_j+d_{jk}x_i-c_{ij}c_{ik}\cdot x_id_{jk}.$$

Note: Note that non-degeneracy conditions ensure associativity of multiplication, defined by the relations. It is also proved, that they are necessary and sufficient to guarantee the PBW property of an algebra, defined via `C_ij` and `D_ij` as above.

Theorem (properties of G-algebras)

Let A be a G-algebra. Then

A has a PBW (Poincaré-Birkhoff-Witt) basis,

A is left and right noetherian,

A is an integral domain.

Setting up a G-algebra

In order to set up a G–algebra one has to do the following steps:

- define a commutative ring $R = K[x_1, \ldots, x_n]$, equipped with a monomial ordering $<$ (see Section 7.2.7.1 [ring declarations (plural)], page 332).
This provides us with the information on a field K (together with its parameters), variables $\{x_i\}$ and an ordering `<`.
From the sequence of variables we will build a G-algebra with the Poincaré -Birkhoff-Witt (PBW) basis $\{x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n}\}$.
- define strictly $n \times n$ upper triangular matrices (of type `matrix`)
  1. $C = \{c_{ij}, i < j\}$, with nonzero entries c_{ij} of type number (c_{ij} for $i \geq j$ will be ignored).
  2. $D = \{d_{ij}, i < j\}$, with polynomial entries d_{ij} from R (d_{ij} for $i \geq j$ will be ignored).
- Call the initialization function `nc_algebra(C,D)` (see Section 7.3.16 [nc_algebra], page 348) with the data C and D.

Plural does not check automatically whether the non-degeneracy conditions hold but it provides a procedure Section 7.5.20.3 [ndcond], page 564 from the library Section 7.5.20 [nctools_lib], page 562 to check this.

7.4.2 Groebner bases in G-algebras

We follow the notations, used in the Singular Manual (e.g. in Section C.1 [Standard bases], page 774).

For a G–algebra A, we denote by ${}_A\langle g_1, \ldots, g_s \rangle$ the left submodule of a free module A^r, generated by elements $\{g_1, \ldots, g_s\} \subset A^r$.

Let $<$ be a fixed monomial well-ordering on the G-algebra A with the Poincaré-Birkhoff-Witt (PBW) basis $\{x^\alpha = x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n}\}$. For a given free module A^r with the basis $\{e_1, \ldots, e_r\}$, $<$ denotes also a fixed module ordering on the set of monomials $\{x^\alpha e_i \mid \alpha \in \mathbf{N}^n, 1 \leq i \leq r\}$.

Definition

For a set $S \subset A^r$, define $L(S)$ to be the K-vector space, spanned on the leading monomials of elements of S, $L(S) = \oplus\{K x^\alpha e_i \mid \exists s \in S, \mathrm{LM(s)} = x^\alpha e_i\}$.

We call $L(S)$ the **span of leading monomials of S.**

Let $I \subset A^r$ be a left A-submodule. A finite set $G \subset I$ is called **a left Groebner basis** of I if and only if $L(G) = L(I)$, that is for any $f \in I \setminus \{0\}$ there exists a $g \in G$ satisfying $\mathrm{LM}(g) \mid \mathrm{LM}(f)$, i.e., if $\mathrm{LM}(f) = x^\alpha e_i$, then $\mathrm{LM}(f) = x^\beta e_i$ with $\beta_j \leq \alpha_j$, $1 \leq j \leq n$.

Remark: In general non-commutative algorithms are working with global well-orderings only (see Section 7.1 [PLURAL], page 317, Section B.2 [Monomial orderings], page 768 and Section 3.3.3 [Term orderings], page 34), unless we deal with graded commutative algebras via Section 7.6 [Graded commutative algebras (SCA)], page 614.

A Groebner basis $G \subset A^r$ is called **minimal** (or **reduced**) if $0 \notin G$ and if $\mathrm{LM}(g) \notin L(G \setminus \{g\})$ for all $g \in G$. Note, that any Groebner basis can be made minimal by deleting successively those g with $\mathrm{LM}(h) \mid \mathrm{LM}(g)$ for some $h \in G \setminus \{g\}$.

For $f \in A^r$ and $G \subset A^r$ we say that f is **completely reduced with respect to** G if no monomial of f is contained in $L(G)$.

Left Normal Form

A map NF : $A^r \times \{G \mid G$ a (left) Groebner basis$\} \to A^r, (f|G) \mapsto \text{NF}(f|G)$, is called a **(left) normal form** on A^r if for any $f \in A^r$ and any left Groebner basis G the following holds:
(i) $\text{NF}(0|G) = 0$,
(ii) if $\text{NF}(f|G) \neq 0$ then $\text{LM}(g)$ does not divide $\text{LM}(\text{NF}(f|G))$ for all $g \in G$,
(iii) $f - \text{NF}(f|G) \in {}_A\langle G\rangle$.
$\text{NF}(f|G)$ is called a **left normal form of f with respect to G** (note that such a map is not unique).

Remark: As we have already mentioned in the definitions `ideal` and `module` (see Section 7.1 [PLURAL], page 317), by `NF` (or `reduce`) PLURAL understands a left normal form. Note, that `rightNF` from Section 7.5.20 [nctools_lib], page 562 allows to compute a right normal form.

Left ideal membership (plural)

For a left Groebner basis G of I the following holds: $f \in I$ if and only if the left normal form $\text{NF}(f|G) = 0$.

For computing a left Groebner basis `G` of `I`, use Section 7.3.26 [std (plural)], page 360.

For computing a left normal form of `f` with respect to `G`, use Section 7.3.23 [reduce (plural)], page 356.

Right ideal membership (plural)

The right ideal membership is analogous to the left one:

for computing a right Groebner basis `G` of `I`, use Section 7.8.10 [rightstd (letterplace)], page 632 from Section 7.5.20 [nctools_lib], page 562,

for computing a right normal form of `f` with respect to `G`, use Section 7.5.20.11 [rightNF], page 571 from Section 7.5.20 [nctools_lib], page 562.

Two-sided ideal membership (plural)

Let J be a two-sided ideal and T be a two-sided Groebner basis of J.

Then $f \in J$ if and only if the left normal form $\text{NF}(f|T) = 0$.

For computing a two-sided Groebner basis `T` of `J`, use Section 7.3.29 [twostd (plural)], page 363,

for computing a normal form of `f` with respect to `T`, use Section 7.3.23 [reduce (plural)], page 356.

7.4.3 Syzygies and resolutions (plural)

Syzygies

Let A be a GR-algebra. A **left** (resp. **right**) **syzygy** between k elements $\{f_1, \ldots, f_k\} \subset A^r$ is a k-tuple $(g_1, \ldots, g_k) \in A^k$ satisfying

$$\sum_{i=1}^{k} g_i f_i = 0 \quad \text{resp.} \quad \sum_{i=1}^{k} f_i g_i = 0.$$

The set of all left (resp. right) syzygies between $\{f_1, ..., f_k\}$ is a left (resp. right) submodule S of A^k.

Remark: With respect to the definitions of `ideal` and `module` (see Section 7.1 [PLURAL], page 317), by `syz` PLURAL understands an inquiry to compute the left syzygy module.

Note, that `rightModulo(M,std(0))` from Section 7.5.20 [nctools_lib], page 562 computes the right syzygy module of `M`.

If `S` is a matrix of a left syzygy module of left submodule given by matrix `M`, then `transpose(S)*transpose(M) = 0` (but, in general, $M \cdot S \neq 0$).

Note, that the syzygy modules of I depend on a choice of generators $\{g_1, \ldots, g_s\}$, but one can show that they depend on I uniquely up to direct summands.

Free resolutions

Let $I = {}_A\langle g_1, \ldots, g_s \rangle \subseteq A^r$ and $M = A^r/I$. A **free resolution of** M is a long exact sequence

$$\ldots \longrightarrow F_2 \xrightarrow{B_2} F_1 \xrightarrow{B_1} F_0 \longrightarrow M \longrightarrow 0,$$

with $\texttt{transpose}(B_{i+1}) \cdot \texttt{transpose}(B_i) = 0$

and where the columns of the matrix B_1 generate I . Note, that resolutions over factor-algebras need not to be of finite length.

Generalized Hilbert Syzygy Theorem

For a G-algebra A, generated by n variables, there exists a free resolution of length smaller or equal than n .

Example:

```
ring R=0,(x,y,z),dp;
matrix d[3][3];
d[1,2]=-z;  d[1,3]=2x;  d[2,3]=-2y;
def U=nc_algebra(1,d); // this algebra is U(sl_2)
setring U;
option(redSB); option(redTail);
ideal I=x3,y3,z3-z;
I=std(I);
I;
↦ I[1]=z3-z
↦ I[2]=y3
↦ I[3]=x3
↦ I[4]=y2z2-y2z
↦ I[5]=x2z2+x2z
↦ I[6]=x2y2z-2xyz2-2xyz+2z2+2z
resolution resI = mres(I,0);
resI;
↦  1      5      7      3
↦ U <--  U <--  U <--  U
↦
↦ 0      1      2      3
↦
list l = resI;
// The matrix A_1 is given by
print(matrix(l[1]));
```

```
↦ z3-z,y3,x3,y2z2-y2z,x2z2+x2z
// We see that the columns of A_1 generate I.
// The matrix A_2 is given by
print(matrix(l[2]));
↦ 0,        0,        y2,  x2,  6yz,        -36xy+18z+24,-6xz,
↦ z2+11z+30,0,        0,   0,   2x2z+12x2,  2x3,         0,
↦ 0,        z2-11z+30,0,   0,   0,          -2y3,        2y2z-12y2,
↦ -y,       0,        -z-5,0,   x2y-6xz-30x,9x2,         x3,
↦ 0,        -x,       0,   -z+5,-y3,        -9y2,        -xy2-4yz+28y
ideal tst; // now let us show that the resolution is exact
matrix TST;
TST = transpose(matrix(l[3]))*transpose(matrix(l[2])); // 2nd term
size(ideal(TST));
↦ 0
TST = transpose(matrix(l[2]))*transpose(matrix(l[1])); // 1st term
size(ideal(TST));
↦ 0
```

7.4.4 References (plural)

The Centre for Computer Algebra Kaiserslautern publishes a series of preprints which are electronically available at https://www.mathematik.uni-kl.de/organisation/zca/reports-on-ca/. Other sources to check are the following books and articles:

Text books

- [DK] Y. Drozd and V. Kirichenko. Finite dimensional algebras. With an appendix by Vlastimil Dlab. Springer, 1994
- [GPS] Greuel, G.-M. and Pfister, G. with contributions by Bachmann, O. ; Lossen, C. and Schönemann, H. A SINGULAR Introduction to Commutative Algebra. Springer, 2002
- [BGV] Bueso, J.; Gomez Torrecillas, J.; Verschoren, A. Algorithmic methods in non-commutative algebra. Applications to quantum groups. Kluwer Academic Publishers, 2003
- [Kr] Kredel, H. Solvable polynomial rings. Shaker, 1993 http://krum.rz.uni-mannheim.de/kredel/kredel_solvable_polynomial_rings.pdf
- [Li] Huishi Li. Non-commutative Gröbner bases and filtered-graded transfer. Springer, 2002
- [MR] McConnell, J.C. and Robson, J.C. Non-commutative Noetherian rings. With the cooperation of L. W. Small. Graduate Studies in Mathematics. 30. Providence, RI: American Mathematical Society (AMS)., 2001

Descriptions of algorithms and problems

- J. Apel. Gröbnerbasen in nichtkommutativen algebren und ihre anwendung. Dissertation, Universität Leipzig, 1988.
- Apel, J. Computational ideal theory in finitely generated extension rings. Theor. Comput. Sci.(2000), 244(1-2):1-33
- O. Bachmann and H. Schönemann. Monomial operations for computations of Gröbner bases. In Reports On Computer Algebra 18. Centre for Computer Algebra, University of Kaiserslautern (1998)
- D. Decker and D. Eisenbud. Sheaf algorithms using the exterior algebra. In Eisenbud, D.; Grayson, D.; Stillman, M.; Sturmfels, B., editor, Computations in algebraic geometry with Macaulay 2, (2001)

- Jose L. Bueso, J. Gomez Torrecillas and F. J. Lobillo. Computing the Gelfand-Kirillov dimension II. In A. Granja, J. A. Hermida and A. Verschoren eds. Ring Theory and Algebraic Geometry, Lect. Not. in Pure and Appl. Maths., Marcel Dekker, 2001.
- Jose L. Bueso, J. Gomez Torrecillas and F. J. Lobillo. Re-filtering and exactness of the Gelfand-Kirillov dimension. Bulletin des Sciences Mathematiques 125(8), 689-715 (2001).
- J. Gomez Torrecillas and F.J. Lobillo. Global homological dimension of multifiltered rings and quantized enveloping algebras. J. Algebra, 225(2):522-533, 2000.
- A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. J. Symbolic Computation, 9(1):1-26, 1990.
- [L1] Levandovskyy, V. PBW Bases, Non-degeneracy Conditions and Applications. In Buchweitz, R.-O. and Lenzing, H., editor, Proceedings of the ICRA X conference, Toronto, 2003.
- [LS] Levandovskyy V.; Schönemann, H. Plural - a computer algebra system for noncommutative polynomial algebras. In Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'03). ACM Press, 2003.
- [LV] Levandovskyy, V. Non-commutative Computer Algebra for polynomial algebras: Gröbner bases, applications and implementation. Doctoral Thesis, Universität Kaiserslautern, 2005. Available online at `http://kluedo.ub.uni-kl.de/volltexte/2005/1883/`.
- [L2] Levandovskyy, V. On preimages of ideals in certain non-commutative algebras. In Pfister G., Cojocaru S. and Ufnarovski, V. (editors), Computational Commutative and Non-Commutative Algebraic Geometry, IOS Press, 2005.
- Mora, T. Gröbner bases for non-commutative polynomial rings. Proc. AAECC 3 Lect. N. Comp. Sci, 229: 353-362, 1986.
- Mora, T. An introduction to commutative and non-commutative Groebner bases. Theor. Comp. Sci., 134: 131-173, 1994.
- T. Nüßler and H. Schönemann. Gröbner bases in algebras with zero-divisors. Preprint 244, Universität Kaiserslautern, 1993. `https://www.mathematik.uni-kl.de/organisation/zca/reports-on-ca/`.
- Schönemann, H. SINGULAR in a Framework for Polynomial Computations. In Joswig, M. and Takayama, N., editor, Algebra, Geometry and Software Systems, pages 163-176. Springer, 2003.
- T. Yan. The geobucket data structure for polynomials. J. Symbolic Computation, 25(3):285-294, March 1998.

7.5 PLURAL libraries

The content of libraries, created for PLURAL is described in the following subsections.

Use the `LIB` command for loading of single libraries.

Note: For any computation in PLURAL, the monomial ordering must be a global ordering.

See also Section D.11.3 [jacobson_lib], page 1977 for the diagonalization of matrices over Ore Euclidean domains.

7.5.1 bimodules_lib

Library: bimodules.lib

Purpose: Tools for handling bimodules

Authors: Ann Christina Foldenauer, Christina.Foldenauer@rwth-aachen.de
Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview:

The main purpose of this library is the handling of bimodules
which will help e.g. to determine weak normal forms of representation matrices
and total divisors within non-commutative, non-simple G-algebras.
We will use modules homomorphisms between a G-algebra and its enveloping algebra
in order to work left Groebner basis theory on bimodules.
Assume we have defined a (non-commutative) G-algebra A over the field K, and an (A,A)-bimodule M.
Instead of working with M over A, we define the enveloping algebra A^{env} = A otimes_K A^{opp}
(this can be done with command envelope(A)) and embed M into A^{env} via imap().
Thus we obtain the left A^{env}-module M otimes 1 in A^{env}.
This has a lot of advantages, because left module theory has much more commands
that are already implemented in SINGULAR:PLURAL. Two important procedures that we can use are std()
which computes the left Groebner basis, and NF() which computes the left normal form.
With the help of this method we are also able to determine the set of bisyzygies of a bimodule.

A built-in command `twostd` in PLURAL computes the two-sided Groebner basis of an ideal
by using the right completion algorithm of [2]. `bistd` from this library uses very different
approach, which is often superior to the right completion.

References:

The procedure bistd() is the implementation of an algorithm M. del Socorro Garcia Roman presented in [1](page 66-78).
[1] Maria del Socorro Garcia Roman, Effective methods in Algebras with PBW bases:
G-algebras and Yang-Baxter Algebras, Ph.D. thesis, Universidad de La Laguna, 2005.
[2] Viktor Levandovskyy, Non-commutative Computer Algebra for polynomial Algebras:
Groebner Bases, Applications and Implementations, Ph.D. thesis, Kaiserlautern, 2005.
[3] N. Jacobson, The theory of rings, AMS, 1943.
[4] P. M. Cohn, Free Rings and their Relations, Academic Press Inc. (London) Ltd., 1971.

Procedures: See also: Section 7.5.10 [ncalg_lib], page 464; Section 7.5.20 [nctools_lib], page 562.

7.5.1.1 bistd

Procedure from library `bimodules.lib` (see Section 7.5.1 [bimodules_lib], page 370).

Usage: bistd(M); M is (two-sided) ideal/module

Return: ideal or module (same type as the argument)

Purpose: Computes the two-sided Groebner basis of an ideal/module with the help the enveloping algebra of the basering, alternative to twostd() for ideals.

Example:

```
LIB "bimodules.lib";
ring w = 0,(x,s),Dp;
def W=nc_algebra(1,s); // 1st shift algebra
setring W;
matrix m[3][3]=[s^2,s+1,0],[s+1,0,s^3-x^2*s],[2*s+1, s^3+s^2, s^2];
print(m);
↦ s2,    s+1,     s+1,
↦ 0,     -x2s+s3,2s+1,
↦ s3+s2,s2,      0
module L = m; module M2 = bistd(L);
print(M2);
↦ 1,1,s+1,
↦ 0,1,0,
↦ 0,0,s2
```

7.5.1.2 bitrinity

Procedure from library `bimodules.lib` (see Section 7.5.1 [bimodules_lib], page 370).

Usage: bitrinity(M); M is (two-sided) ideal/module

Return: ring, the enveloping algebra of the basering, with objects in it. additionally it exports a list L = Coeff, Bisyz.

Theory: Let psi_s be the epimorphism of left R (X) R^{opp} modules:
psi_s(s (X)_K t) = smt := (s_1 m t_1, ... , s_s m t_s) = (\psi(s_1 (X) t_1) , \dots , psi(s_s (X) t_s)) in R^s.
Then psi_s(A) := (psi_s(a_{ij})) for every matrix A in Mat(n x m, R)$.
For a two-sided ideal I = < f_1, ... , f_j> with Groebner basis G = {g_1, ... , g_k} in R, Coeff is the Coefficient-Matrix and BiSyz a bisyzygy matrix.
Let C be the submatrix of Coeff, where C is Coeff without the first row. Then (g_1,...,g_k) = psi_s(C^T * (f_1 ... f_j)^T) and (0,...,0) = psi_s(BiSyz^T * (f_1 ... f_j)^T).
The first row of Coeff (G_1 ... G_n)$ corresponds to the image of the Groebner basis of I: psi_s((G_1 ... G_n)) = G = {g_1 ... g_k }.
For a (R,R)-bimodule M with Groebner basis G = {g_1, ... , g_k} in R^r, Coeff is the coefficient matrix and BiSyz a bisyzygy matrix.
Let C be the submatrix of Coeff, where C is Coeff without the first r rows. Then (g_1 ... g_k) = psi_s(C^T * (f_1 ... f_j)^T) and (0 ... 0) = psi_s(BiSyz^T * (f_1 ... f_j)^T).
The first r rows of Coeff = (G_1 ... G_n) (Here G_i denotes to the i-th column of the first r rows) corresponds to the image of the Groebner basis of M: psi_s((G_1 ... G_n)) = G = {g_1 ... g_k}.

Purpose: This procedure returns a coefficient matrix in the enveloping algebra of the basering R, that gives implicitly the two-sided Groebner basis of a (R,R)-bimodule M and the coefficients that produce the Groebner basis with the help of the originally used generators of M. Additionally it calculates the bisyzygies of M as left-module of the enveloping algebra of R.

Auxiliary procedures:

Note: To get list L = Coeff, BiSyz, we set: def G = bitrinity(); setring G; L; or $L[1]; L[2];.

Example:

```
LIB "bimodules.lib";
ring r = 0,(x,s),dp;
```

```
def R = nc_algebra(1,s); setring R; // 1st shift algebra
poly f = x*s + s^2; // only one generator
ideal I = f; // note, two sided Groebner basis of I is xs, s^2
def G = bitrinity(I);
setring G;
print(L[1]); // Coeff
↦ S2, SX,
↦ s-S,-s+S+1
//the first row shows the Groebnerbasis of I consists of
// psi_s(SX) = xs , phi(S^2) = s^2:
// remember phi(a (X) b - c (X) d) = psi_s(a (X) b) - phi(c (X) d) := ab - cd in R.
// psi_s((-s+S+1)*(x*s + s^2)) = psi_s(-xs2-s3+xsS+xs+s2S)
// = -xs^2-s^3+xs^2+xs+s^3 = xs
// psi_s((s-S)*(x*s + s^2)) = psi_s(xs2+s3-xsS-s2S+s2) = s^2
print(L[2]);  //Bisyzygies
↦ sX+sS-2s-SX-S2,x+s-X-S+1,s2-2sS+S2
// e.g. psi_s((x2-2sS+s-X2+2S2+2X+S-1)(x*s + s^2))
// = psi_s(x3s+x2s2-2xs2S+xs2-2s3S+s3-xsX2+2xsS2+2xsX+xsS-xs-s2X2+2s2S2+2s2X-s2S)
// = x^3s+x^2s^2-2xs^3+xs^2-2s^4+s^3-xsx^2+2xs^3+2xsx+xs^2-xs-s^2x^2+2s^4+2s^2x-s^3
// = 0 in R
```

7.5.1.3 liftenvelope

Procedure from library `bimodules.lib` (see Section 7.5.1 [bimodules_lib], page 370).

Usage: liftenvelope(M,g); M ideal/module, g poly

Return: ring, the enveloping algebra of the basering R.
Given a two-sided ideal M in R and a polynomial g in R this procedure returns the enveloping algebra of R. Additionally it exports a list l = C, B; where B is the left Groebner basis of the left-syzygies of M \otimes 1 and C is a vector of coefficients in the enveloping algebra of R such that psi_s(C^T *(f_1 \dots f_n)) = g.
psi_s is an epimorphism of left R (X) R^{opp} modules:
psi_s (s (X)_K t) = smt := (s_1 m t_1, ... , s_s m t_s) = (\psi(s_1 (X) t_1) , \dots , psi(s_s (X) t_s)) in R^s.
Then psi_s(A) := (psi_s(a_{ij})) for every matrix A in Mat(n x m, R)$.

Assume: The second component has to be an element of the first component.

Purpose: This procedure is used for computing total divisors. Let {f_1, ..., f_n} be the generators of the first component and let the second component be called g. Then the returned list l = C, B = (b_1, ..., b_n); defines an affine set A = C + sum_i a_i b_i with (a_1,..,a_n) in the enveloping algebra of the basering R such that psi_s(a^T * (f_1 ... f_n)) = g for all a in A. For certain rings R, we csn find pure tensors within this set A, and if we do, liftenvelope() helps us to decide whether f is a total divisor of g.

Note: To get list l = C, B. we set: def G = liftenvelope(); setring G; l; or l[1]; l[2];.

Example:

```
LIB "bimodules.lib";
ring r = 0,(x,s),dp;
def R = nc_algebra(1,s); setring R;
ideal I = x*s;
poly p = s*x*s*x;  // = (s (x) x) * x*s = (sX) * x*s
p;
```

```
↦ x2s2+3xs2+2s2
def J = liftenvelope(I,p);
setring J;
print(l[1]);
↦ 0
//2s+SX = (2s (x) 1) + (1 (x) sx)
print(l[2]);
↦ sX-2s-SX,x-X+1,s2-2sS+S2
// Groebnerbasis of BiSyz(I) as LeftSyz in R^{env}
// We get : 2s+SX + ( sX - 2s -SX) = sX  - a pure tensor!!!!
```

7.5.1.4 CompDecomp

Procedure from library `bimodules.lib` (see Section 7.5.1 [bimodules_lib], page 370).

Usage: CompDecomp(p); p poly

Note: This procedure only works if the basering is an enveloping algebra A^{env} of a (noncommutative) ring A. Thus also the polynomial in the argument has to be in A^{env}.

Return: Returns an ideal I in A^{env}, where the sum of all terms of the argument with the same right side (of the tensor summands) are stored as a generator of I.
Let b != c, then for p = (a (X) b) + (c (X) b) + (a (X) c) the ideal I := CompDecomp(p) is given by: I[1] = (a (X) b) + (c (X) b); I[2] = a (X) c.

Purpose: By decomposing the polynomial we can easily check whether the given polynomial is a pure tensor.

Example:

```
LIB "bimodules.lib";
ring r = 0,(x,s),dp;
def R = nc_algebra(1,s); setring R; //1st shift algebra
def Re = envelope(R); setring Re; //basering is now R^{env} = R (X) R^{opp}
poly f = X*S*x^2+5*x*S*X+S*X; f;
↦ x2SX+x2S+5xSX+SX
ideal I = CompDecomp(f);
print(matrix(I)); // what means that f = (x2+5x+1)*SX + x2*S
↦ x2SX+5xSX+SX,x2S
poly p = x*S+X^2*S+2*s+x*X^2*s+5*x*s; p;
↦ xsX2+5xs+xS+2s+SX2+2SX+S
ideal Q = CompDecomp(p);
print(matrix(Q));
↦ xsX2,5xs+2s,xS+S,SX2,2SX
```

7.5.1.5 isPureTensor

Procedure from library `bimodules.lib` (see Section 7.5.1 [bimodules_lib], page 370).

Usage: isPureTensor(g); g poly

Note: This procedure only works if the basering is an enveloping algebra A^{env} of a (noncommutative) ring A. Thus also the polynomial in the argument has to be in A^{env}.

Return: Returns 0 if g is not a pure tensor and if g is a pure tensor then isPureTensor() returns a vector v with v = a*gen(1)+b*gen(2) = (a,b)^T with a (X) b = g.

Purpose: Checks whether a given polynomial in $\A^{env}$ is a pure tensor. This is also an auxiliary procedure for checking total divisibility.

Example:

```
LIB "bimodules.lib";
ring r = 0,(x,s),dp;
def R = nc_algebra(1,s); setring R; //1st shift algebra
def Re = envelope(R); setring Re; //basering is now R^{env} = R (X) R^{opp}
poly p = x*(x*s)*x + s^2*x; p;
↦ x3s+x2s+xs2+2s2
// p is of the form q(X)1, a pure tensor indeed:
isPureTensor(p);
↦ x3s*gen(1)+x2s*gen(1)+xs2*gen(1)+2s2*gen(1)+gen(2)
// v = transpose( x3s+x2s+xs2+2s2  1 ) i.e. p = x3s+x2s+xs2+2s2 (X) 1
poly g = S*X+ x*s*X+ S^2*x;
g;
↦ xsX+xS2+SX
isPureTensor(g); // indeed g is not a pure tensor
↦ 0
poly d = x*X+s*X+x*S*X+s*S*X;d;
↦ xSX+xX+sSX+sX
isPureTensor(d); // d is a pure tensor indeed
↦ x*gen(1)+s*gen(1)+SX*gen(2)+X*gen(2)
// v = transpose( x+s  S*X+X ) i.e. d = x+s (X) s*x+x
// remember that * denotes to the opposite mulitiplication s*x = xs in R.
```

7.5.1.6 isTwoSidedGB

Procedure from library bimodules.lib (see Section 7.5.1 [bimodules_lib], page 370).

Usage: isTwoSidedGB(I); I ideal

Return: Returns 0 if the generators of a given ideal are not two-sided, 1 if they are.\\

Note: This procedure should only be used for non-commutative rings, as every element is two-sided in a commutative ring.

Purpose: Auxiliary procedure for diagonal forms. Let R be a non-commutative ring (e.g. G-algebra), and p in R, this program checks whether p is two-sided i.e. Rp = pR.

Example:

```
LIB "bimodules.lib";
ring r = 0,(x,s),dp;
def R = nc_algebra(1,s); setring R; //1st shift algebra
ideal I = s^2, x*s, s^2 + 3*x*s;
isTwoSidedGB(I); // I is two-sided
↦ 1
ideal J = s^2+x;
isTwoSidedGB(J); // J is not two-sided; twostd(J) = s,x;
↦ 0
```

7.5.2 bfun_lib

Library: bfun.lib

Purpose: Algorithms for b-functions and Bernstein-Sato polynomial

Authors: Daniel Andres, daniel.andres@math.rwth-aachen.de
Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview: Given a polynomial ring R = K[x_1,...,x_n] and a polynomial F in R, one is interested in the global b-function (also known as Bernstein-Sato polynomial) b(s) in K[s], defined to be the non-zero monic polynomial of minimal degree, satisfying a functional identity L * F^{s+1} = b(s) F^s, for some operator L in D[s] (* stands for the action of differential operator)
By D one denotes the n-th Weyl algebra
K<x_1,...,x_n,d_1,...,d_n | d_j x_j = x_j d_j +1>.
One is interested in the following data:
- Bernstein-Sato polynomial b(s) in K[s],
- the list of its roots, which are known to be rational
- the multiplicities of the roots.

There is a constructive definition of a b-function of a holonomic ideal I in D (that is, an ideal I in a Weyl algebra D, such that D/I is holonomic module) with respect to the given weight vector w: For a polynomial p in D, its initial form w.r.t. (-w,w) is defined as the sum of all terms of p which have maximal weighted total degree where the weight of x_i is -w_i and the weight of d_i is w_i. Let J be the initial ideal of I w.r.t. (-w,w), i.e. the K-vector space generated by all initial forms w.r.t (-w,w) of elements of I. Put s = w_1 x_1 d_1 + ... + w_n x_n d_n. Then the monic generator b_w(s) of the intersection of J with the PID K[s] is called the b-function of I w.r.t. w. Unlike Bernstein-Sato polynomial, general b-function with respect to arbitrary weights need not have rational roots at all. However, b-function of a holonomic ideal is known to be non-zero as well.

References:
[SST] Saito, Sturmfels, Takayama: Groebner Deformations of Hypergeometric Differential Equations (2000),
Noro: An Efficient Modular Algorithm for Computing the Global b-function, (2002).

Procedures: See also: Section 7.5.4 [dmod_lib], page 400; Section 7.5.5 [dmodapp_lib], page 420; Section 7.5.7 [dmodvar_lib], page 453; Section D.6.13 [gmssing_lib], page 1702.

7.5.2.1 bfct

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: bfct(f [,s,t,v]); f a poly, s,t optional ints, v an optional intvec

Return: list of ideal and intvec

Purpose: computes the roots of the Bernstein-Sato polynomial b(s)
for the hypersurface defined by f.

Assume: The basering is commutative and of characteristic 0.

Background:
In this proc, the initial Malgrange ideal is computed according to
the algorithm by Masayuki Noro and then a system of linear equations is
solved by linear reductions.

Note: In the output list, the ideal contains all the roots
and the intvec their multiplicities.

If s<>0, `std` is used for GB computations,
otherwise, and by default, `slimgb` is used.
If t<>0, a matrix ordering is used for Groebner basis computations,
otherwise, and by default, a block ordering is used.
If v is a positive weight vector, v is used for homogenization
computations, otherwise and by default, no weights are used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
bfct(f);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
intvec v = 3,2;
bfct(f,1,0,v);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
```

7.5.2.2 bfctSyz

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: bfctSyz(f [,r,s,t,u,v]); f poly, r,s,t,u optional ints, v opt. intvec

Return: list of ideal and intvec

Purpose: computes the roots of the Bernstein-Sato polynomial b(s)
for the hypersurface defined by f

Assume: The basering is commutative and of characteristic 0.

Background:
In this proc, the initial Malgrange ideal is computed according to
the algorithm by Masayuki Noro and then a system of linear equations is
solved by computing syzygies.

Note: In the output list, the ideal contains all the roots and the intvec
their multiplicities.
If r<>0, `std` is used for GB computations in characteristic 0,
otherwise, and by default, `slimgb` is used.
If s<>0, a matrix ordering is used for GB computations, otherwise,
and by default, a block ordering is used.
If t<>0, the computation of the intersection is solely performed over
charasteristic 0, otherwise and by default, a modular method is used.

If u<>0 and by default, `std` is used for GB computations in characteristic >0, otherwise, `slimgb` is used.
If v is a positive weight vector, v is used for homogenization computations, otherwise and by default, no weights are used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
bfctSyz(f);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
intvec v = 3,2;
bfctSyz(f,0,1,1,0,v);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
```

7.5.2.3 bfctAnn

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: bfctAnn(f [,a,b,c]); f a poly, a, b, c optional ints

Return: list of ideal and intvec

Purpose: computes the roots of the Bernstein-Sato polynomial b(s) for the hypersurface defined by f.

Assume: The basering is commutative and of characteristic 0.

Background:
In this proc, Ann(f^s) is computed and then a system of linear equations is solved by linear reductions.

Note: In the output list, the ideal contains all the roots and the intvec their multiplicities.
If a<>0, only f is appended to Ann(f^s), otherwise, and by default, f and all its partial derivatives are appended.
If b<>0, `std` is used for GB computations, otherwise, and by default, `slimgb` is used.
If c<>0, `std` is used for Groebner basis computations of ideals <I+J> when I is already a Groebner basis of <I>.
Otherwise, and by default the engine determined by the switch b is used.
Note that in the case c<>0, the choice for b will be overwritten only for the types of ideals mentioned above.
This means that if b<>0, specifying c has no effect.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
bfctAnn(f);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
def R = reiffen(4,5); setring R;
RC; // the Reiffen curve in 4,5
↦ xy4+y5+x4
bfctAnn(RC,0,1);
↦ [1]:
↦    _[1]=-9/20
↦    _[2]=-11/20
↦    _[3]=-13/20
↦    _[4]=-7/10
↦    _[5]=-17/20
↦    _[6]=-9/10
↦    _[7]=-19/20
↦    _[8]=-1
↦    _[9]=-21/20
↦    _[10]=-11/10
↦    _[11]=-23/20
↦    _[12]=-13/10
↦    _[13]=-27/20
↦ [2]:
↦    1,1,1,1,1,1,1,1,1,1,1,1,1
```

7.5.2.4 bfctOneGB

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: bfctOneGB(f [,s,t]); f a poly, s,t optional ints

Return: list of ideal and intvec

Purpose: computes the roots of the Bernstein-Sato polynomial b(s) for the
hypersurface defined by f, using only one GB computation

Assume: The basering is commutative and of characteristic 0.

Background:
In this proc, the initial Malgrange ideal is computed based on the
algorithm by Masayuki Noro and combined with an elimination ordering.

Note: In the output list, the ideal contains all the roots and the intvec
their multiplicities.
If s<>0, `std` is used for the GB computation, otherwise,
and by default, `slimgb` is used.

If t<>0, a matrix ordering is used for GB computations,
otherwise, and by default, a block ordering is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
bfctOneGB(f);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
bfctOneGB(f,1,1);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
```

7.5.2.5 bfctIdeal

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: bfctIdeal(I,w[,s,t]); I an ideal, w an intvec, s,t optional ints

Return: list of ideal and intvec

Purpose: computes the roots of the global b-function of I w.r.t. the weight (-w,w).

Assume: The basering is the n-th Weyl algebra in characteristic 0 and for all
1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the
sequence of variables is given by x(1),...,x(n),D(1),...,D(n),
where D(i) is the differential operator belonging to x(i).
Further we assume that I is holonomic.

Background:

In this proc, the initial ideal of I is computed according to the
algorithm by Masayuki Noro and then a system of linear equations is
solved by linear reductions.

Note: In the output list, say L,
- L[1] of type ideal contains all the rational roots of a b-function,
- L[2] of type intvec contains the multiplicities of above roots,
- optional L[3] of type string is the part of b-function without rational roots.
Note, that a b-function of degree 0 is encoded via L[1][1]=0, L[2]=0 and
L[3] is 1 (for nonzero constant) or 0 (for zero b-function).
If s<>0, `std` is used for GB computations in characteristic 0,
otherwise, and by default, `slimgb` is used.
If t<>0, a matrix ordering is used for GB computations, otherwise,
and by default, a block ordering is used.

Display: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring @D = 0,(x,y,Dx,Dy),dp;
def D = Weyl();
setring D;
ideal I = 3*x^2*Dy+2*y*Dx,2*x*Dx+3*y*Dy+6; I = std(I);
intvec w1 = 0,1;
intvec w2 = 2,3;
bfctIdeal(I,w1);
↦ [1]:
↦    _[1]=0
↦    _[2]=-2/3
↦    _[3]=-4/3
↦ [2]:
↦    1,1,1
bfctIdeal(I,w2,0,1);
↦ [1]:
↦    _[1]=-6
↦ [2]:
↦    1
ideal J = I[size(I)]; // J is not holonomic by construction
bfctIdeal(J,w1); //  b-function of D/J w.r.t. w1 is non-zero
↦ WARNING: given ideal is not holonomic
↦ ... setting bound for degree of b-function to 10 and proceeding
↦ [1]:
↦    _[1]=0
↦    _[2]=-2/3
↦    _[3]=-4/3
↦ [2]:
↦    1,1,1
bfctIdeal(J,w2); //  b-function of D/J w.r.t. w2 is zero
↦ WARNING: given ideal is not holonomic
↦ ... setting bound for degree of b-function to 10 and proceeding
↦ // Intersection is zero
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    0
↦ [3]:
↦    0
```

7.5.2.6 pIntersect

Procedure from library bfun.lib (see Section 7.5.2 [bfun_lib], page 375).

Usage: pIntersect(f, I [,s]); f a poly, I an ideal, s an optional int

Return: vector, coefficient vector of the monic polynomial

Purpose: compute the intersection of ideal I with the subalgebra K[f]

Assume: I is given as Groebner basis, basering is not a qring.

Note: If the intersection is zero, this proc might not terminate.
If s>0 is given, it is searched for the generator of the intersection
only up to degree s. Otherwise (and by default), no bound is assumed.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
def D = initialMalgrange(f);
setring D;
inF;
↦ inF[1]=x*Dt
↦ inF[2]=2*x*y*Dx+3*y^2*Dx-y^2*Dy-2*x*Dy
↦ inF[3]=2*x^2*Dx+x*y*Dx+x*y*Dy+18*t*Dt+9*x*Dx-x*Dy+6*y*Dy+4*x+18
↦ inF[4]=18*t*Dt^2+6*y*Dt*Dy-y*Dt+27*Dt
↦ inF[5]=y^2*Dt
↦ inF[6]=2*t*y*Dt+2*x*y*Dx+2*y^2*Dx-6*t*Dt-3*x*Dx-x*Dy-2*y*Dy+2*y-6
↦ inF[7]=x*y^2+y^3+x^2
↦ inF[8]=2*y^3*Dx-2*y^3*Dy-3*y^2*Dx-2*x*y*Dy+y^2*Dy-4*y^2+36*t*Dt+18*x*Dx+1\
   2*y*Dy+36
pIntersect(t*Dt,inF);
↦ gen(4)-1/36*gen(2)
pIntersect(t*Dt,inF,1);
↦ // Try a bound of at least 2
↦ 0
```

7.5.2.7 pIntersectSyz

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: pIntersectSyz(f, I [,p,s,t]); f poly, I ideal, p,t optial ints, p prime

Return: vector, coefficient vector of the monic polynomial

Purpose: compute the intersection of an ideal I with the subalgebra K[f]

Assume: I is given as Groebner basis.

Note: If the intersection is zero, this procedure might not terminate.
If p>0 is given, this proc computes the generator of the intersection in char p first and then only searches for a generator of the obtained degree in the basering. Otherwise, it searches for all degrees by computing syzygies.
If s<>0, `std` is used for Groebner basis computations in char 0, otherwise, and by default, `slimgb` is used.
If t<>0 and by default, `std` is used for Groebner basis computations in char >0, otherwise, `slimgb` is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
def D = initialMalgrange(f);
setring D;
inF;
↦ inF[1]=x*Dt
↦ inF[2]=2*x*y*Dx+3*y^2*Dx-y^2*Dy-2*x*Dy
↦ inF[3]=2*x^2*Dx+x*y*Dx+x*y*Dy+18*t*Dt+9*x*Dx-x*Dy+6*y*Dy+4*x+18
↦ inF[4]=18*t*Dt^2+6*y*Dt*Dy-y*Dt+27*Dt
↦ inF[5]=y^2*Dt
↦ inF[6]=2*t*y*Dt+2*x*y*Dx+2*y^2*Dx-6*t*Dt-3*x*Dx-x*Dy-2*y*Dy+2*y-6
↦ inF[7]=x*y^2+y^3+x^2
↦ inF[8]=2*y^3*Dx-2*y^3*Dy-3*y^2*Dx-2*x*y*Dy+y^2*Dy-4*y^2+36*t*Dt+18*x*Dx+1\
   2*y*Dy+36
poly s = t*Dt;
pIntersectSyz(s,inF);
↦ gen(4)-1/36*gen(2)
int p = prime(20000);
pIntersectSyz(s,inF,p,0,0);
↦ gen(4)-1/36*gen(2)
```

7.5.2.8 linReduce

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: linReduce(f, I [,s,t,u]); f a poly, I an ideal, s,t,u optional ints

Return: poly or list, linear reductum (over field) of f by elements from I

Purpose: reduce a polynomial only by linear reductions (no monomial multiplications)

Note: If s<>0, a list consisting of the reduced polynomial and the coefficient vector of the used reductions is returned, otherwise (and by default) only reduced polynomial is returned.
If t<>0 (and by default) all monomials are reduced (if possible), otherwise, only leading monomials are reduced.
If u<>0 (and by default), the ideal is linearly pre-reduced, i.e. instead of the given ideal, the output of `linReduceIdeal` is used.
If u is set to 0 and the given ideal does not equal the output of `linReduceIdeal`, the result might not be as expected.

Display: If `printlevel`=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
ideal I = 1,y,xy;
poly f = 5xy+7y+3;
poly g = 7x+5y+3;
linReduce(g,I);     // reduces tails
↦ 7x
linReduce(g,I,0,0); // no reductions of tails
↦ 7x+5y+3
```

```
linReduce(f,I,1);   // reduces tails and shows reductions used
↦ [1]:
↦    0
↦ [2]:
↦    -5*gen(3)-7*gen(2)-3*gen(1)
f = x3+y2+x2+y+x;
I = x3-y3, y3-x2,x3-y2,x2-y,y2-x;
list l = linReduce(f,I,1);
l;
↦ [1]:
↦    5y
↦ [2]:
↦    gen(5)-4*gen(4)+2*gen(3)-3*gen(2)-3*gen(1)
module M = I;
f - (l[1]-(M*l[2])[1,1]);
↦ 0
```

7.5.2.9 linReduceIdeal

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: linReduceIdeal(I [,s,t,u]); I an ideal, s,t,u optional ints

Return: ideal or list, linear reductum (over field) of I by its elements

Purpose: reduces a list of polys only by linear reductions (no monomial multiplications)

Note: If s<>0, a list consisting of the reduced ideal and the coefficient vectors of the used reductions given as module is returned.
Otherwise (and by default), only the reduced ideal is returned.
If t<>0 (and by default) all monomials are reduced (if possible), otherwise, only leading monomials are reduced.
If u<>0 (and by default), the ideal is first sorted in increasing order.
If u is set to 0 and the given ideal is not sorted in the way described, the result might not be as expected.

Display: If `printlevel`=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
ideal I = 3,x+9,y4+5x,2y4+7x+2;
linReduceIdeal(I);     // reduces tails
↦ _[1]=0
↦ _[2]=3
↦ _[3]=x
↦ _[4]=y4
linReduceIdeal(I,0,0); // no reductions of tails
↦ _[1]=0
↦ _[2]=3
↦ _[3]=x+9
↦ _[4]=y4+5x
list l = linReduceIdeal(I,1); // reduces tails and shows reductions used
```

```
l;
↦ [1]:
↦    _[1]=0
↦    _[2]=3
↦    _[3]=x
↦    _[4]=y4
↦ [2]:
↦    _[1]=gen(4)-2*gen(3)+3*gen(2)-29/3*gen(1)
↦    _[2]=gen(1)
↦    _[3]=gen(2)-3*gen(1)
↦    _[4]=gen(3)-5*gen(2)+15*gen(1)
module M = I;
matrix(l[1]) - matrix(M)*matrix(l[2]);
↦ _[1,1]=0
↦ _[1,2]=0
↦ _[1,3]=0
↦ _[1,4]=0
```

7.5.2.10 linSyzSolve

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: linSyzSolve(I[,s]); I an ideal, s an optional int

Return: vector, coefficient vector of linear combination of 0 in elements of I

Purpose: compute a linear dependency between the elements of an ideal
if such one exists

Note: If s<>0, `std` is used for Groebner basis computations,
otherwise, `slimgb` is used.
By default, `slimgb` is used in char 0 and `std` in char >0.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "bfun.lib";
ring r = 0,x,dp;
ideal I = x,2x;
linSyzSolve(I);
↦ gen(2)-2*gen(1)
ideal J = x,x2;
linSyzSolve(J);
↦ 0
```

7.5.2.11 allPositive

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: allPositive(v); v an intvec

Return: int, 1 if all components of v are positive, or 0 otherwise

Purpose: check whether all components of an intvec are positive

Example:

```
LIB "bfun.lib";
intvec v = 1,2,3;
allPositive(v);
↦ 1
intvec w = 1,-2,3;
allPositive(w);
↦ 0
```

7.5.2.12 scalarProd

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: scalarProd(v,w); v,w intvecs

Return: int, the standard scalar product of v and w

Purpose: computes the scalar product of two intvecs

Assume: the arguments are of the same size

Example:

```
LIB "bfun.lib";
intvec v = 1,2,3;
intvec w = 4,5,6;
scalarProd(v,w);
↦ 32
```

7.5.2.13 vec2poly

Procedure from library `bfun.lib` (see Section 7.5.2 [bfun_lib], page 375).

Usage: vec2poly(v [,i]); v a vector or an intvec, i an optional int

Return: poly, an univariate polynomial in i-th variable with coefficients given by v

Purpose: constructs an univariate polynomial in K[var(i)] with given coefficients, such that the coefficient at var(i)^{j-1} is v[j].

Note: The optional argument i must be positive, by default i is 1.

Example:

```
LIB "bfun.lib";
ring r = 0,(x,y),dp;
vector v = gen(1) + 3*gen(3) + 22/9*gen(4);
intvec iv = 3,2,1;
vec2poly(v,2);
↦ 22/9y3+3y2+1
vec2poly(iv);
↦ x2+2x+3
```

7.5.3 central_lib

Library: central.lib

Purpose: Computation of central elements of GR-algebras

Author: Oleksandr Motsak, U@D, where U={motsak}, D={mathematik.uni-kl.de}

Overview: A library for computing elements of the center and centralizers of sets of elements in GR-algebras.

Procedures:

7.5.3.1 centralizeSet

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centralizeSet(F, V); F, V ideals

Input: F, V finite sets of elements of the base algebra

Return: ideal, generated by computed elements

Purpose: computes a vector space basis of the centralizer of the set F in the vector space generated by V over the ground field

Example:

```
LIB "central.lib";
ring A = 0,(e(1..4)),dp;
matrix D[4][4]=0;
D[2,4] = -e(1);
D[3,4] = -e(2);
// This is A_4_1 - the first real Lie algebra of dimension 4.
def A_4_1 = nc_algebra(1,D); setring A_4_1;
ideal F = variablesSorted(); F;
↦ F[1]=e(1)
↦ F[2]=e(4)
↦ F[3]=e(3)
↦ F[4]=e(2)
// the center of A_4_1 is generated by
// e(1) and -1/2*e(2)^2+e(1)*e(3)
// therefore one may consider computing it in the following way:
// 1. Compute a PBW basis consisting of
//    monomials with exponent <= (1,2,1,0)
ideal V = PBW_maxMonom( e(1) * e(2)^ 2 * e(3) );
// 2. Compute the centralizer of F within the vector space
//    spanned by these monomials:
ideal C = centralizeSet( F, V ); C;
↦ C[1]=e(1)
↦ C[2]=e(2)^2-2*e(1)*e(3)
inCenter(C); // check the result
↦ 1
```

See also: Section 7.5.3.7 [centralizer], page 391; Section 7.5.3.2 [centralizerVS], page 387; Section 7.5.3.11 [inCentralizer], page 393.

7.5.3.2 centralizerVS

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centralizerVS(F, D); F ideal, D int

Return: ideal, generated by computed elements

Purpose: computes a vector space basis of centralizer(F) up to degree D

Note: D must be non-negative

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
```

```
matrix D[3][3]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
ideal F = x, y;
// find generators of the vector space of elements
// of degree <= 4 commuting with x and y:
ideal C = centralizerVS(F, 4);
C;
↦ C[1]=4xy+z2-2z
↦ C[2]=16x2y2+8xyz2+z4-32xyz-4z3-4z2+16z
inCentralizer(C, F); // check the result
↦ 1
```

See also: Section 7.5.3.4 [centerVS], page 388; Section 7.5.3.7 [centralizer], page 391; Section 7.5.3.11 [inCentralizer], page 393.

7.5.3.3 centralizerRed

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centralizerRed(F, D[, N]); F ideal, D int, N optional int

Return: ideal, generated by computed elements

Purpose: computes subalgebra generators of centralizer(F) up to degree D.

Note: In general, one cannot compute the whole centralizer(F).
Hence, one has to specify a termination condition via arguments D and/or N.
If D is positive, only centralizing elements up to degree D are computed.
If D is negative, the termination is determined by N only.
If N is given, the computation stops if at least N elements have been found.
Warning: if N is given and bigger than the actual number of generators,
the procedure may not terminate.
Current ordering must be a degree compatible well-ordering.

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
ideal F = x, y;
// find subalgebra generators of degree <= 4 of the subalgebra of
// all elements commuting with x and y:
ideal C = centralizerRed(F, 4);
C;
↦ C[1]=4xy+z2-2z
inCentralizer(C, F); // check the result
↦ 1
```

See also: Section 7.5.3.5 [centerRed], page 389; Section 7.5.3.7 [centralizer], page 391; Section 7.5.3.2 [centralizerVS], page 387; Section 7.5.3.11 [inCentralizer], page 393.

7.5.3.4 centerVS

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centerVS(D); D int

Return: ideal, generated by computed elements

Purpose: computes a vector space basis of the center up to degree D

Note: D must be non-negative

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// find a basis of the vector space of all
// central elements of degree <= 4:
ideal Z = centerVS(4);
Z;
↦ Z[1]=4xy+z2-2z
↦ Z[2]=16x2y2+8xyz2+z4-32xyz-4z3-4z2+16z
// note that the second element is the square of the first
// plus a multiple of the first:
Z[2] - Z[1]^2 + 8*Z[1];
↦ 0
inCenter(Z); // check the result
↦ 1
```

See also: Section 7.5.3.6 [center], page 390; Section 7.5.3.2 [centralizerVS], page 387; Section 7.5.3.10 [inCenter], page 393.

7.5.3.5 centerRed

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centerRed(D[, N]); D int, N optional int

Return: ideal, generated by computed elements

Purpose: computes subalgebra generators of the center up to degree D

Note: In general, one cannot compute the whole center.
Hence, one has to specify a termination condition via arguments D and/or N.
If D is positive, only central elements up to degree D will be found.
If D is negative, the termination is determined by N only.
If N is given, the computation stops if at least N elements have been found.
Warning: if N is given and bigger than the actual number of generators,
the procedure may not terminate.
Current ordering must be a degree compatible well-ordering.

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=z;
def A = nc_algebra(1,D); setring A; // it is a Heisenberg algebra
// find a basis of the vector space of
// central elements of degree <= 3:
```

```
ideal VSZ = centerVS(3);
// There should be 3 degrees of z.
VSZ;
↦ VSZ[1]=z
↦ VSZ[2]=z2
↦ VSZ[3]=z3
inCenter(VSZ); // check the result
↦ 1
// find "minimal" central elements of degree <= 3
ideal SAZ = centerRed(3);
// Only 'z' must be computed
SAZ;
↦ SAZ[1]=z
inCenter(SAZ); // check the result
↦ 1
```

See also: Section 7.5.3.6 [center], page 390; Section 7.5.3.4 [centerVS], page 388; Section 7.5.3.3 [centralizerRed], page 388; Section 7.5.3.10 [inCenter], page 393.

7.5.3.6 center

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: center(D[, N]); D int, N optional int

Return: ideal, generated by computed elements

Purpose: computes subalgebra generators of the center up to degree D

Note: In general, one cannot compute the whole center.
Hence, one has to specify a termination condition via arguments D and/or N.
If D is positive, only central elements up to degree D will be found.
If D is negative, the termination is determined by N only.
If N is given, the computation stops if at least N elements have been found.
Warning: if N is given and bigger than the actual number of generators,
the procedure may not terminate.
Current ordering must be a degree compatible well-ordering.

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z,t),dp;
matrix D[4][4]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2) tensored with K[t]
// find generators of the center of degree <= 3:
ideal Z = center(3);
Z;
↦ Z[1]=t
↦ Z[2]=4xy+z2-2z
inCenter(Z); // check the result
↦ 1
// find at least one generator of the center:
ideal Z2 = center(-1, 1);
Z2;
↦ Z2[1]=t
inCenter(Z2); // check the result
```

```
↦ 1
```

See also: Section 7.5.3.7 [centralizer], page 391; Section 7.5.3.10 [inCenter], page 393.

7.5.3.7 centralizer

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: centralizer(F, D[, N]); F poly/ideal, D int, N optional int

Return: ideal, generated by computed elements

Purpose: computes subalgebra generators of centralizer(F) up to degree D

Note: In general, one cannot compute the whole centralizer(F).
Hence, one has to specify a termination condition via arguments D and/or N.
If D is positive, only centralizing elements up to degree D will be found.
If D is negative, the termination is determined by N only.
If N is given, the computation stops if at least N elements have been found.
Warning: if N is given and bigger than the actual number of generators,
the procedure may not terminate.
Current ordering must be a degree compatible well-ordering.

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z; D[1,3]=2*x; D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
poly f = 4*x*y+z^2-2*z; // a central polynomial
f;
↦ 4xy+z2-2z
// find generators of the centralizer of f of degree <= 2:
ideal c = centralizer(f, 2);
c;  // since f is central, the answer consists of generators of A
↦ c[1]=z
↦ c[2]=y
↦ c[3]=x
inCentralizer(c, f); // check the result
↦ 1
// find at least two generators of the centralizer of f:
ideal cc = centralizer(f,-1,2);
cc;
↦ cc[1]=z
↦ cc[2]=y
↦ cc[3]=x
inCentralizer(cc, f); // check the result
↦ 1
poly g = z^2-2*z; // some non-central polynomial
// find generators of the centralizer of g of degree <= 2:
c = centralizer(g, 2);
c;
↦ c[1]=z
↦ c[2]=xy
inCentralizer(c, g); // check the result
↦ 1
```

```
// find at least one generator of the centralizer of g:
centralizer(g,-1,1);
↦ _[1]=z
// find at least two generators of the centralizer of g:
cc = centralizer(g,-1,2);
cc;
↦ cc[1]=z
↦ cc[2]=xy
inCentralizer(cc, g); // check the result
↦ 1
```

See also: Section 7.5.3.6 [center], page 390; Section 7.5.3.11 [inCentralizer], page 393.

7.5.3.8 sa_reduce

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: sa_reduce(V); V ideal

Return: ideal, generated by computed elements

Purpose: compute a subalgebra basis of an algebra generated by the elements of V

Note: At the moment the usage of this procedure is limited to G-algebras

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z; D[1,3]=2*x; D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
poly f = 4*x*y+z^2-2*z; // a central polynomial
ideal I = f, f*f, f*f*f - 10*f*f, f+3*z^3; I;
↦ I[1]=4xy+z2-2z
↦ I[2]=16x2y2+8xyz2+z4-32xyz-4z3+32xy+4z2
↦ I[3]=64x3y3+48x2y2z2+12xyz4+z6-288x2y2z-96xyz3-6z5+352x2y2+224xyz2+2z4-12\
   8xyz+32z3-64xy-40z2
↦ I[4]=3z3+4xy+z2-2z
sa_reduce(I); // should be just f and z^3
↦ _[1]=4xy+z2-2z
↦ _[2]=z3
```

See also: Section 7.5.3.9 [sa_poly_reduce], page 392.

7.5.3.9 sa_poly_reduce

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: sa_poly_reduce(p, V); p poly, V ideal

Return: polynomial, a reduction of p w.r.t. V

Purpose: computes a reduction of the polynomial p w.r.t. the subalgebra generated by elements of V

Note: At the moment the usage of this procedure is limited to G-algebras

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z; D[1,3]=2*x; D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
poly f = 4*x*y+z^2-2*z; // a central polynomial
sa_poly_reduce(f + 3*f*f + x, ideal(f) ); // should be just 'x'
↦ x
```

See also: Section 7.5.3.8 [sa_reduce], page 392.

7.5.3.10 inCenter

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: inCenter(E); E poly/list/ideal

Return: integer, 1 if E is in the center, 0 otherwise

Purpose: check whether the elements of E are central

Example:

```
LIB "central.lib";
ring R=0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;
D[1,3]=2*x;
D[2,3]=-2*y;
def r = nc_algebra(1,D); setring r; // this is U(sl_2)
poly p=4*x*y+z^2-2*z;
inCenter(p);
↦ 1
poly f=4*x*y;
inCenter(f);
↦ 0
list l= list( 1, p, p^2, p^3);
inCenter(l);
↦ 1
ideal I= p, f;
inCenter(I);
↦ 0
```

7.5.3.11 inCentralizer

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: inCentralizer(E, S); E poly/list/ideal, S poly/ideal

Return: integer, 1 if E is in the centralizer(S), 0 otherwise

Purpose: check whether the elements of E are in the centralizer(S)

Example:

```
LIB "central.lib";
ring R = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;
```

```
def r = nc_algebra(1,D); setring r; // the Heisenberg algebra
poly f = x^2;
poly a = z; // 'z' is central => it lies in every centralizer!
poly b = y^2;
inCentralizer(a, f);
↦ 1
inCentralizer(b, f);
↦ 0
list  l = list(1, a);
inCentralizer(l, f);
↦ 1
ideal I = a, b;
inCentralizer(I, f);
↦ 0
printlevel = 2;
inCentralizer(a, f); // yes
↦ 1
inCentralizer(b, f); // no
↦ [1]:
↦    POLY: y2 is NOT in the centralizer of polynomial {x2}
↦ 0
```

7.5.3.12 isCartan

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: isCartan(f); f poly

Purpose: check whether f is a Cartan element.

Return: integer, 1 if f is a Cartan element and 0 otherwise.

Note: f is a Cartan element of the algebra A
if and only if for all g in A there exists C in K such that [f, g] = C * g
if and only if for all variables v_i there exist C in K such that [f, v_i] = C * v_i.

Example:

```
LIB "central.lib";
ring R=0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;
D[1,3]=2*x;
D[2,3]=-2*y;
def r = nc_algebra(1,D); setring r; // this is U(sl_2) with cartan - z
isCartan(z); // yes!
↦ 1
poly p=4*x*y+z^2-2*z;
isCartan(p); // central elements are Cartan elements!
↦ 1
poly f=4*x*y;
isCartan(f); // no way!
↦ 0
isCartan( 10 + p + z ); // scalar + central + cartan
↦ 1
```

7.5.3.13 applyAdF

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: applyAdF(B, f); B ideal, f poly

Purpose: Apply Ad_f to every element of B

Return: ideal, generated by Ad_f(B[i]), 1<=i<=size(B)

Note: Ad_f(v) := [f, v] = f*v - v*f

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// Let us consider the linear map Ad_{e} from A_2 into A.
// Compute the PBW basis of A_2:
ideal Basis = PBW_maxDeg( 2 ); Basis;
↦ Basis[1]=e
↦ Basis[2]=f
↦ Basis[3]=h
↦ Basis[4]=h2
↦ Basis[5]=fh
↦ Basis[6]=f2
↦ Basis[7]=eh
↦ Basis[8]=ef
↦ Basis[9]=e2
// Compute images of basis elements under the linear map Ad_e:
ideal Image = applyAdF( Basis, e ); Image;
↦ Image[1]=0
↦ Image[2]=h
↦ Image[3]=-2e
↦ Image[4]=-4eh-4e
↦ Image[5]=-2ef+h2+2h
↦ Image[6]=2fh-2f
↦ Image[7]=-2e2
↦ Image[8]=eh
↦ Image[9]=0
// Now we have a linear map given by: Basis_i --> Image_i
// Let's compute its kernel K:
// 1. compute syzygy module C:
module C = linearMapKernel( Image ); C;
↦ C[1]=gen(1)
↦ C[2]=gen(8)+1/4*gen(4)-1/2*gen(3)
↦ C[3]=gen(9)
// 2. compute corresponding combinations of basis vectors:
ideal K = linearCombinations(Basis, C); K;
↦ K[1]=e
↦ K[2]=ef+1/4h2-1/2h
↦ K[3]=e2
// Let's check that Ad_e(K) is zero:
applyAdF( K, e );
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
```

See also: Section 7.5.3.14 [linearMapKernel], page 396.

7.5.3.14 linearMapKernel

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: linearMapKernel(Images); Images ideal

Purpose: Computes the syzygy module of the linear map given by Images.

Return: syzygy module, or int(0) if all images are zeroes

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
def A = nc_algebra(1,D); // this algebra is U(sl_2)
setring A;
// Let us consider the linear map Ad_{e} from A_2 into A.
// Compute the PBW basis of A_2:
ideal Basis = PBW_maxDeg( 2 ); Basis;
↦ Basis[1]=e
↦ Basis[2]=f
↦ Basis[3]=h
↦ Basis[4]=h2
↦ Basis[5]=fh
↦ Basis[6]=f2
↦ Basis[7]=eh
↦ Basis[8]=ef
↦ Basis[9]=e2
// Compute images of basis elements under the linear map Ad_e:
ideal Image = applyAdF( Basis, e ); Image;
↦ Image[1]=0
↦ Image[2]=h
↦ Image[3]=-2e
↦ Image[4]=-4eh-4e
↦ Image[5]=-2ef+h2+2h
↦ Image[6]=2fh-2f
↦ Image[7]=-2e2
↦ Image[8]=eh
↦ Image[9]=0
// Now we have a linear map given by: Basis_i --> Image_i
// Let's compute its kernel K:
// 1. compute syzygy module C:
module C = linearMapKernel( Image ); C;
↦ C[1]=gen(1)
↦ C[2]=gen(8)+1/4*gen(4)-1/2*gen(3)
↦ C[3]=gen(9)
// 2. compute corresponding combinations of basis vectors:
ideal K = linearCombinations(Basis, C); K;
↦ K[1]=e
↦ K[2]=ef+1/4h2-1/2h
↦ K[3]=e2
// Let's check that Ad_e(K) is zero:
ideal Z = applyAdF( K, e ); Z;
↦ Z[1]=0
↦ Z[2]=0
↦ Z[3]=0
// Now linearMapKernel will return a single integer 0:
def CC  = linearMapKernel(Z); typeof(CC); CC;
↦ int
↦ 0
```

See also: Section 7.5.3.13 [applyAdF], page 395; Section 7.5.3.14 [linearMapKernel], page 396.

7.5.3.15 linearCombinations

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: linearCombinations(Basis, C); Basis ideal, C module

Purpose: forms linear combinations of elements from Basis by replacing gen(i) by Basis[i] in C

Return: ideal generated by computed linear combinations

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// Let us consider the linear map Ad_{e} from A_2 into A.
// Compute the PBW basis of A_2:
ideal Basis = PBW_maxDeg( 2 ); Basis;
↦ Basis[1]=e
↦ Basis[2]=f
↦ Basis[3]=h
↦ Basis[4]=h2
↦ Basis[5]=fh
↦ Basis[6]=f2
↦ Basis[7]=eh
↦ Basis[8]=ef
↦ Basis[9]=e2
// Compute images of basis elements under the linear map Ad_e:
ideal Image = applyAdF( Basis, e ); Image;
↦ Image[1]=0
↦ Image[2]=h
↦ Image[3]=-2e
↦ Image[4]=-4eh-4e
↦ Image[5]=-2ef+h2+2h
↦ Image[6]=2fh-2f
↦ Image[7]=-2e2
↦ Image[8]=eh
↦ Image[9]=0
// Now we have a linear map given by: Basis_i --> Image_i
// Let's compute its kernel K:
// 1. compute syzygy module C:
module C = linearMapKernel( Image ); C;
```

```
↦ C[1]=gen(1)
↦ C[2]=gen(8)+1/4*gen(4)-1/2*gen(3)
↦ C[3]=gen(9)
// 2. compute corresponding combinations of basis vectors:
ideal K = linearCombinations(Basis, C); K;
↦ K[1]=e
↦ K[2]=ef+1/4h2-1/2h
↦ K[3]=e2
// Let's check that Ad_e(K) is zero:
applyAdF( K, e );
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
```

See also: Section 7.5.3.13 [applyAdF], page 395; Section 7.5.3.14 [linearMapKernel], page 396.

7.5.3.16 variablesStandard

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: variablesStandard();

Return: ideal, generated by algebra variables

Purpose: computes the set of algebra variables taken in their natural order

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// Variables in their natural order:
variablesStandard();
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
```

See also: Section 7.5.3.17 [variablesSorted], page 398.

7.5.3.17 variablesSorted

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: variablesSorted();

Return: ideal, generated by sorted algebra variables

Purpose: computes the set of algebra variables sorted so that
Cartan variables go first

Note: This is a heuristics for the computation of the center:
it is better to compute centralizers of Cartan variables first since in this
case we can omit solving the system of equations.

Example:

```
LIB "central.lib";
ring AA = 0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;  D[1,3]=2*x;  D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// There is only one Cartan variable - z in U(sl_2),
// it must go 1st:
variablesSorted();
↦ _[1]=z
↦ _[2]=y
↦ _[3]=x
```

See also: Section 7.5.3.16 [variablesStandard], page 398.

7.5.3.18 PBW_eqDeg

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: PBW_eqDeg(Deg); Deg int

Purpose: Compute PBW elements of a given degree.

Return: ideal consisting of found elements.

Note: Unit is omitted. Weights are ignored!

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
// PBW Basis of A_2 \ A_1 - monomials of degree == 2:
PBW_eqDeg( 2 );
↦ _[1]=h2
↦ _[2]=fh
↦ _[3]=f2
↦ _[4]=eh
↦ _[5]=ef
↦ _[6]=e2
```

7.5.3.19 PBW_maxDeg

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: PBW_maxDeg(MaxDeg); MaxDeg int

Purpose: Compute PBW elements up to a given maximal degree.

Return: ideal consisting of found elements.

Note: unit is omitted. Weights are ignored!

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
```

```
def A = nc_algebra(1,D); // this algebra is U(sl_2)
setring A;
// PBW Basis of A_2 - monomials of degree <= 2, without unit:
PBW_maxDeg( 2 );
↦ _[1]=e
↦ _[2]=f
↦ _[3]=h
↦ _[4]=h2
↦ _[5]=fh
↦ _[6]=f2
↦ _[7]=eh
↦ _[8]=ef
↦ _[9]=e2
```

7.5.3.20 PBW_maxMonom

Procedure from library `central.lib` (see Section 7.5.3 [central_lib], page 386).

Usage: PBW_maxMonom(m); m poly

Purpose: Compute PBW elements up to a given maximal one.

Return: ideal consisting of found elements.

Note: Unit is omitted. Weights are ignored!

Example:

```
LIB "central.lib";
ring AA = 0,(e,f,h),dp;
matrix D[3][3]=0;
D[1,2]=-h;  D[1,3]=2*e;  D[2,3]=-2*f;
def A = nc_algebra(1,D); // this algebra is U(sl_2)
setring A;
// At most 1st degree in e, h and at most 2nd degree in f, unit is omitted:
PBW_maxMonom( e*(f^2)* h );
↦ _[1]=e
↦ _[2]=f
↦ _[3]=ef
↦ _[4]=f2
↦ _[5]=ef2
↦ _[6]=h
↦ _[7]=eh
↦ _[8]=fh
↦ _[9]=efh
↦ _[10]=f2h
↦ _[11]=ef2h
```

7.5.4 dmod_lib

Library: dmod.lib

Purpose: Algorithms for algebraic D-modules

Authors: Viktor Levandovskyy, levandov@math.rwth-aachen.de
Jorge Martin Morales, jorge@unizar.es

Overview: Let K be a field of characteristic 0. Given a polynomial ring
R = K[x_1,...,x_n] and a polynomial F in R,
one is interested in the R[1/F]-module of rank one, generated by
the symbol F^s for a symbolic discrete variable s.
In fact, the module R[1/F]*F^s has a structure of a D(R)[s]-module, where D(R)
is an n-th Weyl algebra K<x_1,...,x_n,d_1,...,d_n | d_j x_j = x_j d_j +1> and
D(R)[s] = D(R) tensored with K[s] over K.
Constructively, one needs to find a left ideal I = I(F^s) in D(R), such
that K[x_1,...,x_n,1/F]*F^s is isomorphic to D(R)/I as a D(R)-module.
We often write just D for D(R) and D[s] for D(R)[s].
One is interested in the following data:
- Ann F^s = I = I(F^s) in D(R)[s], denoted by LD in the output
- global Bernstein polynomial in K[s], denoted by bs,
- its minimal integer root s0, the list of all roots of bs, which are known
to be rational, with their multiplicities, which is denoted by BS
- Ann F^s0 = I(F^s0) in D(R), denoted by LD0 in the output
(LD0 is a holonomic ideal in D(R))
- Ann^(1) F^s in D(R)[s], denoted by LD1 (logarithmic derivations)
- an operator in D(R)[s], denoted by PS, such that the functional equality
PS*F^(s+1) = bs*F^s holds in K[x_1,...,x_n,1/F]*F^s.

References:

We provide the following implementations of algorithms:
(OT) the classical Ann F^s algorithm from Oaku and Takayama (Journal of
Pure and Applied Math., 1999),
(LOT) Levandovskyy's modification of the Oaku-Takayama algorithm (ISSAC 2007)
(BM) the Ann F^s algorithm by Briancon and Maisonobe (Remarques sur
l'ideal de Bernstein associe a des polynomes, preprint, 2002)
(LM08) V. Levandovskyy and J. Martin-Morales, ISSAC 2008
(C) Countinho, A Primer of Algebraic D-Modules,
(SST) Saito, Sturmfels, Takayama 'Groebner Deformations of Hypergeometric
Differential Equations', Springer, 2000

Guide:
- Ann F^s = I(F^s) = LD in D(R)[s] can be computed by Sannfs [BM, OT, LOT]
- Ann^(1) F^s in D(R)[s] can be computed by Sannfslog
- global Bernstein polynomial bs in K[s] can be computed by bernsteinBM
- Ann F^s0 = I(F^s0) = LD0 in D(R) can be computed by annfs0, annfs, annfsBM,
annfsOT, annfsLOT, annfs2, annfsRB etc.
- all the relevant data to F^s (LD, LD0, bs, PS) are computed by operatorBM
- operator PS can be computed via operatorModulo or operatorBM

- annihilator of F^{s1} for a number s1 is computed with annfspecial
- annihilator of F_1^s_1 * ... * F_p^s_p is computed with annfsBMI
- computing the multiplicity of a rational number r in the Bernstein poly
of a given ideal goes with checkRoot
- check, whether a given univariate polynomial divides the Bernstein poly
goes with checkFactor

Procedures: See also: Section 7.5.2 [bfun_lib], page 375; Section 7.5.5 [dmodapp_lib], page 420; Section 7.5.7 [dmodvar_lib], page 453; Section D.6.13 [gmssing_lib], page 1702.

7.5.4.1 annfs

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfs(f [,S,eng]); f a poly, S a string, eng an optional int

Return: ring

Purpose: compute the D-module structure of basering[1/f]*f^s with the algorithm given in S and with the Groebner basis engine given in "eng"

Note: activate the output ring with the `setring` command.
String S; S can be one of the following:
'bm' (default) - for the algorithm of Briancon and Maisonobe,
'ot' - for the algorithm of Oaku and Takayama,
'lot' - for the Levandovskyy's modification of the algorithm of OT.
If eng <>0, `std` is used for Groebner basis computations,
otherwise and by default `slimgb` is used.
In the output ring:
- the ideal LD (which is a Groebner basis) is the needed D-module structure,
- the list BS contains roots and multiplicities of a BS-polynomial of f.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = z*x^2+y^3;
def A  = annfs(F); // here, the default BM algorithm will be used
setring A; // the Weyl algebra in (x,y,z,Dx,Dy,Dz)
LD; //the annihilator of F^{-1} over A
↦ LD[1]=y*Dy+3*z*Dz+3
↦ LD[2]=x*Dx-2*z*Dz
↦ LD[3]=x^2*Dy-3*y^2*Dz
↦ LD[4]=3*y^2*Dx-2*x*z*Dy
↦ LD[5]=y^3*Dz+x^2*z*Dz+x^2
↦ LD[6]=2*x*z*Dy^2+9*y*z*Dx*Dz+3*y*Dx
↦ LD[7]=9*y*z*Dx^2*Dz+4*z^2*Dy^2*Dz+3*y*Dx^2+2*z*Dy^2
↦ LD[8]=4*z^2*Dy^3*Dz-27*z^2*Dx^2*Dz^2+2*z*Dy^3-54*z*Dx^2*Dz-6*Dx^2
BS; // roots with multiplicities of BS polynomial
↦ [1]:
↦    _[1]=-1
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-5/6
↦    _[5]=-7/6
↦ [2]:
↦    1,1,1,1,1
```

7.5.4.2 annfspecial

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfspecial(F,n); F a poly, number n

Return: ring

Purpose: compute the annihilator ideal of F^n in the Weyl Algebra
for the given rational number n

Assume: basering is commutative, the number n is rational.

Note: Activate the output ring by `setring` command. In the ring the ideal called `annfalpha` is exported.
We compute the real annihilator for any rational value of n (both
generic and exceptional). The implementation fixes a bug in
the Algorithm 5.3.15 from Saito-Sturmfels-Takayama.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y),dp;
poly F = x3-y2;
bernsteinBM(F); // the roots of Bernstein-Sato poly: -7/6, -1, -5/6
↦ [1]:
↦    _[1]=-1
↦    _[2]=-5/6
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
// *** first example: generic root
def A = annfspecial(F,-5/6);
setring A; print(annfalpha); kill A; setring r;
↦ 2*x*Dx+3*y*Dy+5,
↦ 3*x^2*Dy+2*y*Dx,
↦ 9*x*y*Dy^2-4*y*Dx^2+12*x*Dy,
↦ 27*y^2*Dy^3+8*y*Dx^3+117*y*Dy^2+72*Dy
// *** second example:  exceptional root since its distance to -1 is integer 2
def A = annfspecial(F,1);
setring A;  print(annfalpha); kill A; setring r;
↦ Dx*Dy,
↦ 2*x*Dx+3*y*Dy-6,
↦ Dy^3,
↦ y*Dy^2-Dy,
↦ 3*x*Dy^2+Dx^2,
↦ 3*x^2*Dy+2*y*Dx,
↦ Dx^3+3*Dy^2,
↦ y*Dx^2+3*x*Dy
// *** third example: exceptional root since its distance to -5/6 is integer 1
def A = annfspecial(F,1/6);
setring A;  print(annfalpha); kill A;
↦ 2*x*Dx+3*y*Dy-1,
↦ 3*x^2*Dy+2*y*Dx,
↦ 27*y*Dy^3+8*Dx^3+9*Dy^2,
↦ 9*x*y*Dy^2-4*y*Dx^2-6*x*Dy
```

7.5.4.3 annfspecialOld

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfspecialOld(I,F,mir,n); I an ideal, F a poly, int mir, number n

Return: ideal

Purpose: compute the annihilator ideal of F^n in the Weyl Algebra
for the given rational number n

Assume: The basering is D[s] and contains 's' explicitly as a variable,
the ideal I is the Ann F^s in D[s] (obtained with e.g. SannfsBM),
the integer 'mir' is the minimal integer root of the BS polynomial of F,
and the number n is rational.

Note: We compute the real annihilator for any rational value of n (both generic and exceptional). The implementation goes along the lines of the Algorithm 5.3.15 from Saito-Sturmfels-Takayama, which has a bug. This procedure is correct for integer values of n.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y),dp;
poly F = x3-y2;
def  B = annfs(F);  setring B;
minIntRoot(BS[1],0);
↦ -1
// So, the minimal integer root is -1
setring r;
def  A = SannfsBM(F);
setring A;
poly F = x3-y2;
annfspecialOld(LD,F,-1,3/4); // generic root
↦ _[1]=4*x*Dx+6*y*Dy-9
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=18*x*y*Dy^2-8*y*Dx^2-33*x*Dy
↦ _[4]=54*y^2*Dy^3+16*y*Dx^3+66*x*Dx*Dy-9*y*Dy^2+66*Dy
annfspecialOld(LD,F,-1,-2);  // integer but still generic root
↦ _[1]=2*x*Dx+3*y*Dy+12
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+33*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3-66*x*Dx*Dy+144*y*Dy^2-66*Dy
annfspecialOld(LD,F,-1,1);   // exceptional integer root
↦ _[1]=Dx*Dy
↦ _[2]=2*x*Dx+3*y*Dy-6
↦ _[3]=Dy^3
↦ _[4]=y*Dy^2-Dy
↦ _[5]=3*x*Dy^2+Dx^2
↦ _[6]=3*x^2*Dy+2*y*Dx
↦ _[7]=Dx^3+3*Dy^2
↦ _[8]=y*Dx^2+3*x*Dy
```

7.5.4.4 Sannfs

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: Sannfs(f [,S,eng]); f a poly, S a string, eng an optional int

Return: ring

Purpose: compute the D-module structure of basering[f^s] with the algorithm given in S and with the Groebner basis engine given in eng

Note: activate the output ring with the `setring` command.
The value of a string S can be
'bm' (default) - for the algorithm of Briancon and Maisonobe,
'lot' - for the Levandovskyy's modification of the algorithm of OT,
'ot' - for the algorithm of Oaku and Takayama.
If eng <>0, `std` is used for Groebner basis computations,
otherwise, and by default `slimgb` is used.
In the output ring:
- the ideal LD is the needed D-module structure.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = x^3+y^3+z^3;
printlevel = 0;
def A  = Sannfs(F); // here, the default BM algorithm will be used
setring A;
LD;
↦ LD[1]=z^2*Dy-y^2*Dz
↦ LD[2]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
```

7.5.4.5 Sannfslog

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: Sannfslog(f [,eng]); f a poly, eng an optional int

Return: ring

Purpose: compute the D-module structure of basering[1/f]*f^s

Note: activate the output ring with the `setring` command.
In the output ring D[s], the ideal LD1 is generated by the elements in Ann F^s in D[s], coming from logarithmic derivations.
If eng <>0, `std` is used for Groebner basis computations,
otherwise, and by default `slimgb` is used.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y),Dp;
poly F = x4+y5+x*y4;
printlevel = 0;
```

```
def A  = Sannfslog(F);
setring A;
LD1;
↦ LD1[1]=4*x^2*Dx+5*x*y*Dx+3*x*y*Dy+4*y^2*Dy-16*x*s-20*y*s
↦ LD1[2]=16*x*y^2*Dx+4*y^3*Dx+12*y^3*Dy-125*x*y*Dx-4*x^2*Dy+5*x*y*Dy-100*y^\
   2*Dy-64*y^2*s+500*y*s
```

7.5.4.6 bernsteinBM

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: bernsteinBM(f [,eng]); f a poly, eng an optional int

Return: list (of roots of the Bernstein polynomial b and their multiplicies)

Purpose: compute the global Bernstein-Sato polynomial for a hypersurface,
defined by f, according to the algorithm by Briancon and Maisonobe

Note: If eng <>0, `std` is used for Groebner basis computations,
otherwise, and by default `slimgb` is used.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z,w),Dp;
poly F = x^3+y^3+z^2*w;
printlevel = 0;
bernsteinBM(F);
↦ [1]:
↦    _[1]=-1
↦    _[2]=-2
↦    _[3]=-3/2
↦    _[4]=-5/3
↦    _[5]=-7/3
↦    _[6]=-7/6
↦    _[7]=-11/6
↦ [2]:
↦    1,1,1,1,1,1,1
```

7.5.4.7 bernsteinLift

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: bernsteinLift(I, F [,eng]); I an ideal, F a poly, eng an optional int

Return: list

Purpose: compute the (multiple of) Bernstein-Sato polynomial with lift-like method,
based on the output of Sannfs-like procedure

Note: the output list contains the roots with multiplicities of the candidate
for being Bernstein-Sato polynomial of f.
If eng <>0, `std` is used for Groebner basis computations,
otherwise and by default `slimgb` is used.
If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = x^3+y^3+z^3;
printlevel = 0;
def A = Sannfs(F);   setring A;
LD;
↦ LD[1]=z^2*Dy-y^2*Dz
↦ LD[2]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
poly F = imap(r,F);
list L  = bernsteinLift(LD,F); L;
↦ [1]:
↦    _[1]=-2
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-1
↦ [2]:
↦    1,1,1,2
poly bs = fl2poly(L,"s"); bs; // the candidate for Bernstein-Sato polynomial
↦ s^5+7*s^4+173/9*s^3+233/9*s^2+154/9*s+40/9
```

7.5.4.8 operatorBM

Procedure from library dmod.lib (see Section 7.5.4 [dmod_lib], page 400).

Usage: operatorBM(f [,eng]); f a poly, eng an optional int

Return: ring

Purpose: compute the B-operator and other relevant data for Ann F^s,
using e.g. algorithm by Briancon and Maisonobe for Ann F^s and BS.

Note: activate the output ring with the setring command. In the output ring D[s]
- the polynomial F is the same as the input,
- the ideal LD is the annihilator of f^s in Dn[s],
- the ideal LD0 is the needed D-mod structure, where LD0 = LD|s=s0,
- the polynomial bs is the global Bernstein polynomial of f in the variable s,
- the list BS contains all the roots with multiplicities of the global Bernstein polynomial of f,
- the polynomial PS is an operator in Dn[s] such that PS*f^(s+1) = bs*f^s.
If eng <>0, std is used for Groebner basis computations,
otherwise and by default slimgb is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = x^3+y^3+z^3;
printlevel = 0;
def A = operatorBM(F);
setring A;
```

```
F; // the original polynomial itself
↦ x^3+y^3+z^3
LD; // generic annihilator
↦ LD[1]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[2]=z^2*Dy-y^2*Dz
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
↦ LD[5]=x^3*Dz+y^3*Dz+z^3*Dz-3*z^2*s
↦ LD[6]=x^3*Dy+y^3*Dy+y^2*z*Dz-3*y^2*s
LD0; // annihilator
↦ LD0[1]=x*Dx+y*Dy+z*Dz+6
↦ LD0[2]=z^2*Dy-y^2*Dz
↦ LD0[3]=z^2*Dx-x^2*Dz
↦ LD0[4]=y^2*Dx-x^2*Dy
↦ LD0[5]=x^3*Dz+y^3*Dz+z^3*Dz+6*z^2
↦ LD0[6]=x^3*Dy+y^3*Dy+y^2*z*Dz+6*y^2
bs; // normalized Bernstein poly
↦ s^5+7*s^4+173/9*s^3+233/9*s^2+154/9*s+40/9
BS; // roots and multiplicities of the Bernstein poly
↦ [1]:
↦    _[1]=-2
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-1
↦ [2]:
↦    1,1,1,2
PS; // the operator, s.t. PS*F^{s+1} = bs*F^s mod LD
↦ 2/81*y*z*Dx^3*Dy*Dz-2/81*y*z*Dy^4*Dz-4/81*y^2*Dy^2*Dz^3-2/81*y*z*Dy*Dz^4+\
   2/81*y*Dx^3*Dy*s-2/81*y*Dy^4*s+2/81*z*Dx^3*Dz*s+2/27*z*Dy^3*Dz*s+2/27*y*D\
   y*Dz^3*s-2/81*z*Dz^4*s+2/27*y*Dx^3*Dy-2/27*y*Dy^4+2/27*z*Dx^3*Dz+2/27*z*D\
   y^3*Dz-10/81*y*Dy*Dz^3-2/27*z*Dz^4+1/27*Dx^3*s^2+1/9*Dy^3*s^2+1/9*Dz^3*s^\
   2+5/27*Dx^3*s+11/27*Dy^3*s+11/27*Dz^3*s+20/81*Dx^3+8/27*Dy^3+16/81*Dz^3
reduce(PS*F-bs,LD); // check the property of PS
↦ 0
```

7.5.4.9 operatorModulo

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: operatorModulo(f,I,b); f a poly, I an ideal, b a poly

Return: poly

Purpose: compute the B-operator from the polynomial f,
ideal I = Ann f^s and Bernstein-Sato polynomial b
using modulo i.e. kernel of module homomorphism

Note: The computations take place in the ring, similar to the one
returned by Sannfs procedure.
Note, that operator is not completely reduced wrt Ann f^{s+1}.
If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
```

```
//  LIB "dmod.lib"; option(prot); option(mem);
ring r = 0,(x,y),Dp;
poly F = x^3+y^3+x*y^3;
def A = Sannfs(F); // here we get LD = ann f^s
setring A;
poly F = imap(r,F);
def B = annfs0(LD,F); // to obtain BS polynomial
list BS = imap(B,BS);   poly bs = fl2poly(BS,"s");
poly PS = operatorModulo(F,LD,bs);
LD = groebner(LD);
PS = NF(PS,subst(LD,s,s+1));; // reduction modulo Ann s^{s+1}
↦ // ** _ is no standard basis
size(PS);
↦ 56
lead(PS);
↦ -2/243*y^3*Dx*Dy^3
reduce(PS*F-bs,LD); // check the defining property of PS
↦ 0
```

7.5.4.10 annfsParamBM

Procedure from library dmod.lib (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfsParamBM(f [,eng]); f a poly, eng an optional int

Return: ring

Purpose: compute the generic Ann F^s and exceptional parametric constellations of a polynomial with parametric coefficients with the BM algorithm

Note: activate the output ring with the setring command. In this ring,
- the ideal LD is the D-module structure oa Ann F^s
- the ideal Param contains special parameters as entries
If eng <>0, std is used for Groebner basis computations,
otherwise, and by default slimgb is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = (0,a,b),(x,y),Dp;
poly F = x^2 - (y-a)*(y-b);
printlevel = 0;
def A = annfsParamBM(F);  setring A;
LD;
↦ LD[1]=2*y*Dx+2*x*Dy+(-a-b)*Dx
↦ LD[2]=x^2*Dy-y^2*Dy+(a+b)*y*Dy+2*y*s+(-a*b)*Dy+(-a-b)*s
↦ LD[3]=4*x^2*Dx+4*x*y*Dy+(-2*a-2*b)*x*Dy-8*x*s+(a^2-2*a*b+b^2)*Dx
Param;
↦ Param[1]=(a-b)
setring r;
poly G = x2-(y-a)^2; // try the exceptional value b=a of parameters
def B = annfsParamBM(G); setring B;
LD;
```

```
↦ LD[1]=y*Dx+x*Dy+(-a)*Dx
↦ LD[2]=x*Dx+y*Dy+(-a)*Dy-2*s
↦ LD[3]=x^2*Dy-y^2*Dy+(2*a)*y*Dy+2*y*s+(-a^2)*Dy+(-2*a)*s
Param;
↦ Param[1]=0
```

7.5.4.11 annfsBMI

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfsBMI(F [,eng,met,us]); F an ideal, eng, met, us optional ints

Return: ring

Purpose: compute two kinds of Bernstein-Sato ideals, associated to
f = F[1]*..*F[P], with the multivariate algorithm by Briancon and Maisonobe.

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD is the annihilator of F[1]^s_1*..*F[P]^s_p,
- the list or ideal BS is a Bernstein-Sato ideal of a polynomial f = F[1]*..*F[P].
If eng <>0, `std` is used for Groebner basis computations,
otherwise, and by default `slimgb` is used.
If met <0, the B-Sigma ideal (cf. Castro and Ucha,
'On the computation of Bernstein-Sato ideals', 2005) is computed.
If 0 < met < P, then the ideal B_P (cf. the paper) is computed.
Otherwise, and by default, the ideal B (cf. the paper) is computed.
If us<>0, then syzygies-driven method is used.
If the output ideal happens to be principal, the list of factors
with their multiplicities is returned instead of the ideal.
If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y),Dp;
ideal F = x,y,x+y;
printlevel = 0;
// *1* let us compute the B ideal
def A = annfsBMI(F);    setring A;
LD; // annihilator
↦ LD[1]=x*Dx+y*Dy-s(1)-s(2)-s(3)
↦ LD[2]=x*y*Dy+y^2*Dy-x*s(2)-y*s(2)-y*s(3)
↦ LD[3]=y^2*Dx*Dy-y^2*Dy^2+y*Dy*s(1)-y*Dx*s(2)+2*y*Dy*s(2)-y*Dx*s(3)+y*Dy*s\
   (3)-s(1)*s(2)-s(2)^2-s(2)*s(3)-s(2)
BS; // Bernstein-Sato ideal
↦ [1]:
↦    _[1]=s(1)+1
↦    _[2]=s(2)+1
↦    _[3]=s(3)+1
↦    _[4]=s(1)+s(2)+s(3)+2
↦    _[5]=s(1)+s(2)+s(3)+3
↦    _[6]=s(1)+s(2)+s(3)+4
↦ [2]:
↦    1,1,1,1,1,1
// *2* now, let us compute B-Sigma ideal
```

```
setring r;
def Sigma = annfsBMI(F,0,-1); setring Sigma;
print(matrix(lead(LD))); // compact form of leading
↦ x*Dx,x*y*Dy,y^2*Dx*Dy
//  monomials from the annihilator
BS; // Bernstein-Sato B-Sigma ideal: it is principal,
↦ [1]:
↦    _[1]=s(1)+s(2)+s(3)+2
↦ [2]:
↦    1
// so factors and multiplicities are returned
// *3* and now, let us compute B-i ideal
setring r;
def Bi = annfsBMI(F,0,3); // that is F[3]=x+y is taken
setring Bi;
print(matrix(lead(LD)));   // compact form of leading
↦ x*Dx,x*y*Dy,y^2*Dx*Dy
//  monomials from the annihilator
BS; // the B_3 ideal: it is principal, so factors
↦ [1]:
↦    _[1]=s(3)+1
↦    _[2]=s(1)+s(2)+s(3)+2
↦ [2]:
↦    1,1
// and multiplicities are returned
```

7.5.4.12 checkRoot

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: checkRoot(f,alpha [,S,eng]); poly f, number alpha, string S, int eng

Return: int

Assume: Basering is a commutative ring, alpha is a positive rational number.

Purpose: check whether a negative rational number -alpha is a root of the global Bernstein-Sato polynomial of f and compute its multiplicity,
with the algorithm given by S and with the Groebner basis engine given by eng.

Note: The annihilator of f^s in D[s] is needed and hence it is computed with the algorithm by Briancon and Maisonobe. The value of a string S can be
'alg1' (default) - for the algorithm 1 of [LM08]
'alg2' - for the algorithm 2 of [LM08]
Depending on the value of S, the output of type int is:
'alg1': 1 only if -alpha is a root of the global Bernstein-Sato polynomial
'alg2': the multiplicity of -alpha as a root of the global Bernstein-Sato polynomial of f. If -alpha is not a root, the output is 0.
If eng <>0, `std` is used for Groebner basis computations,
otherwise (and by default) `slimgb` is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
printlevel=0;
ring r = 0,(x,y),Dp;
poly F = x^4+y^5+x*y^4;
checkRoot(F,11/20);    // -11/20 is a root of bf
↦ 1
poly G = x*y;
checkRoot(G,1,"alg2"); // -1 is a root of bg with multiplicity 2
↦ 2
```

7.5.4.13 SannfsBFCT

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: SannfsBFCT(f [,a,b,c]); f a poly, a,b,c optional ints

Return: ring

Purpose: compute a Groebner basis either of Ann(f^s)+<f> or of
Ann(f^s)+<f,f_1,...,f_n> in D[s]

Note: Activate the output ring with the `setring` command.
This procedure, unlike SannfsBM, returns the ring D[s] with an anti-elimination ordering for s.
The output ring contains an ideal `LD`, being a Groebner basis
either of Ann(f^s)+<f>, if a=0 (and by default), or of
Ann(f^s)+<f,f_1,...,f_n>, otherwise.
Here, f_i stands for the i-th partial derivative of f.
If b<>0, `std` is used for Groebner basis computations,
otherwise, and by default `slimgb` is used.
If c<>0, `std` is used for Groebner basis computations of
ideals <I+J> when I is already a Groebner basis of <I>.
Otherwise, and by default the engine determined by the switch b is
used. Note that in the case c<>0, the choice for b will be
overwritten only for the types of ideals mentioned above.
This means that if b<>0, specifying c has no effect.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z,w),Dp;
poly F = x^3+y^3+z^3*w;
// compute Ann(F^s)+<F> using slimgb only
def A = SannfsBFCT(F);
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    s
↦ //        block   2 : ordering dp
↦ //                  : names    x y z w Dx Dy Dz Dw
↦ //        block   3 : ordering C
↦ // noncommutative relations:
```

```
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
↦ //    Dzz=z*Dz+1
↦ //    Dww=w*Dw+1
LD;
↦ LD[1]=z*Dz-3*w*Dw
↦ LD[2]=3*s-x*Dx-y*Dy-3*w*Dw
↦ LD[3]=y^2*Dx-x^2*Dy
↦ LD[4]=z^2*w*Dy-y^2*Dz
↦ LD[5]=z^3*Dy-3*y^2*Dw
↦ LD[6]=z^2*w*Dx-x^2*Dz
↦ LD[7]=z^3*Dx-3*x^2*Dw
↦ LD[8]=z^3*w+x^3+y^3
↦ LD[9]=x^3*Dy+y^3*Dy+3*y^2*w*Dw+3*y^2
↦ LD[10]=x^3*Dx+x^2*y*Dy+3*x^2*w*Dw+3*x^2
↦ LD[11]=3*z*w^2*Dy*Dw-y^2*Dz^2+2*z*w*Dy
↦ LD[12]=3*z*w^2*Dx*Dw-x^2*Dz^2+2*z*w*Dx
↦ LD[13]=3*z^2*w^2*Dw+x^3*Dz+y^3*Dz+3*z^2*w
↦ LD[14]=9*w^3*Dy*Dw^2-y^2*Dz^3+18*w^2*Dy*Dw+2*w*Dy
↦ LD[15]=9*w^3*Dx*Dw^2-x^2*Dz^3+18*w^2*Dx*Dw+2*w*Dx
↦ LD[16]=9*z*w^3*Dw^2+x^3*Dz^2+y^3*Dz^2+24*z*w^2*Dw+6*z*w
↦ LD[17]=27*w^4*Dw^3+x^3*Dz^3+y^3*Dz^3+135*w^3*Dw^2+114*w^2*Dw+6*w
// the Bernstein-Sato poly of F:
vec2poly(pIntersect(s,LD));
↦ s^6+28/3*s^5+320/9*s^4+1910/27*s^3+2093/27*s^2+1198/27*s+280/27
// a fancier example:
def R = reiffen(4,5); setring R;
RC; // the Reiffen curve in 4,5
↦ xy4+y5+x4
// compute Ann(RC^s)+<RC,diff(RC,x),diff(RC,y)>
// using std for GB computations of ideals <I+J>
// where I is already a GB of <I>
// and slimgb for other ideals
def B = SannfsBFCT(RC,1,0,1);
setring B;
// the Bernstein-Sato poly of RC:
(s-1)*vec2poly(pIntersect(s,LD));
↦ s^13+10*s^12+44*s^11+44099/400*s^10+13355001/80000*s^9+22138611/160000*s^\
  8+1747493/160000*s^7-7874303503/64000000*s^6-4244944536107/2560000000*s^\
  5-3066298289417/2560000000*s^4-2787777479229/5120000000*s^3-19980507461\
  787/128000000000*s^2-663659243177931/25600000000000*s-4883920107969/25\
  6000000000000
```

7.5.4.14 annfs0

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfs0(I, F [,eng]); I an ideal, F a poly, eng an optional int

Return: ring

Purpose: compute the annihilator ideal of f^s in the Weyl Algebra, based on the output of Sannfs-like procedure

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD (which is a Groebner basis) is the annihilator of f^s,

- the list BS contains the roots with multiplicities of BS polynomial of f.
If eng <>0, `std` is used for Groebner basis computations,
otherwise and by default `slimgb` is used.
If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = x^3+y^3+z^3;
printlevel = 0;
def A = SannfsBM(F);   setring A;
// alternatively, one can use SannfsOT or SannfsLOT
LD;
↦ LD[1]=z^2*Dy-y^2*Dz
↦ LD[2]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
poly F = imap(r,F);
def B  = annfs0(LD,F);  setring B;
LD;
↦ LD[1]=x*Dx+y*Dy+z*Dz+6
↦ LD[2]=z^2*Dy-y^2*Dz
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
↦ LD[5]=x^3*Dz+y^3*Dz+z^3*Dz+6*z^2
↦ LD[6]=x^3*Dy+y^3*Dy+y^2*z*Dz+6*y^2
BS;
↦ [1]:
↦    _[1]=-2
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-1
↦ [2]:
↦    1,1,1,2
```

7.5.4.15 annfs2

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfs2(I, F [,eng]); I an ideal, F a poly, eng an optional int

Return: ring

Purpose: compute the annihilator ideal of f^s in the Weyl Algebra,
based on the output of Sannfs-like procedure
annfs2 uses shorter expressions in the variable s (the idea of Noro).

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD (which is a Groebner basis) is the annihilator of f^s,
- the list BS contains the roots with multiplicities of the BS polynomial.
If eng <>0, `std` is used for Groebner basis computations,
otherwise and by default `slimgb` is used.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
poly F = x^3+y^3+z^3;
printlevel = 0;
def A = SannfsBM(F);
setring A;
LD;
↦ LD[1]=z^2*Dy-y^2*Dz
↦ LD[2]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
poly F = imap(r,F);
def B  = annfs2(LD,F);
setring B;
LD;
↦ LD[1]=x*Dx+y*Dy+z*Dz+6
↦ LD[2]=z^2*Dy-y^2*Dz
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
↦ LD[5]=x^3*Dz+y^3*Dz+z^3*Dz+6*z^2
↦ LD[6]=x^3*Dy+y^3*Dy+y^2*z*Dz+6*y^2
BS;
↦ [1]:
↦    _[1]=-2
↦    _[2]=-5/3
↦    _[3]=-4/3
↦    _[4]=-1
↦ [2]:
↦    1,1,1,2
```

7.5.4.16 annfsRB

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: annfsRB(I, F [,eng]); I an ideal, F a poly, eng an optional int

Return: ring

Purpose: compute the annihilator ideal of f^s in the Weyl Algebra,
based on the output of Sannfs like procedure

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD (which is a Groebner basis) is the annihilator of f^s,
- the list BS contains the roots with multiplicities of a Bernstein polynomial of f.
If eng <>0, `std` is used for Groebner basis computations,
otherwise and by default `slimgb` is used.
This procedure uses in addition to F its Jacobian ideal.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),Dp;
```

```
poly F = x^3+y^3+z^3;
printlevel = 0;
def A = SannfsBM(F); setring A;
LD; // s-parametric ahhinilator
↦ LD[1]=z^2*Dy-y^2*Dz
↦ LD[2]=x*Dx+y*Dy+z*Dz-3*s
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
poly F = imap(r,F);
def B  = annfsRB(LD,F); setring B;
LD;
↦ LD[1]=x*Dx+y*Dy+z*Dz+6
↦ LD[2]=z^2*Dy-y^2*Dz
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
↦ LD[5]=x^3*Dz+y^3*Dz+z^3*Dz+6*z^2
↦ LD[6]=x^3*Dy+y^3*Dy+y^2*z*Dz+6*y^2
BS;
↦ [1]:
↦    _[1]=-2
↦    _[2]=-5/3
↦    _[3]=-4/3
↦    _[4]=-1
↦ [2]:
↦    1,1,1,2
```

7.5.4.17 checkFactor

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: checkFactor(I,f,qs [,eng]); I an ideal, f a poly, qs a poly, eng an optional int

Assume: checkFactor is called from the basering, created by Sannfs-like proc,
that is, from the Weyl algebra in x1,..,xN,d1,..,dN tensored with K[s].
The ideal I is the annihilator of f^s in D[s], that is the ideal, computed
by Sannfs-like procedure (usually called LD there).
Moreover, f is a polynomial in K[x1,..,xN] and qs is a polynomial in K[s].

Return: int, 1 if qs is a factor of the global Bernstein polynomial of f and 0 otherwise

Purpose: check whether a univariate polynomial qs is a factor of the
Bernstein-Sato polynomial of f without explicit knowledge of the latter.

Note: If eng <>0, `std` is used for Groebner basis computations,
otherwise (and by default) `slimgb` is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y),Dp;
poly F = x^4+y^5+x*y^4;
printlevel = 0;
def A = Sannfs(F);
setring A;
```

```
poly F = imap(r,F);
checkFactor(LD,F,20*s+31);     // -31/20 is not a root of bs
↦ 0
checkFactor(LD,F,20*s+11);     // -11/20 is a root of bs
↦ 1
checkFactor(LD,F,(20*s+11)^2); // the multiplicity of -11/20 is 1
↦ 0
```

7.5.4.18 arrange

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: arrange(p); int p

Return: ring

Purpose: set up the polynomial, describing a hyperplane arrangement

Note: must be executed in a commutative ring

Assume: basering is present and it is commutative

Example:

```
LIB "dmod.lib";
ring X = 0,(x,y,z,t),dp;
poly q = arrange(3);
factorize(q,1);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=x+y
↦ _[4]=z
↦ _[5]=x+z
↦ _[6]=y+z
↦ _[7]=x+y+z
```

7.5.4.19 reiffen

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: reiffen(p, q); int p, int q

Return: ring

Purpose: set up the polynomial, describing a Reiffen curve

Note: activate the output ring with the `setring` command and
find the curve as a polynomial RC.
A Reiffen curve is defined as RC = x^p + y^q + xy^{q-1}, q >= p+1 >= 5

Example:

```
LIB "dmod.lib";
def r = reiffen(4,5);
setring r;
RC;
↦ xy4+y5+x4
```

7.5.4.20 isHolonomic

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: isHolonomic(M); M an ideal/module/matrix

Return: int, 1 if M is holonomic over the base ring, and 0 otherwise

Assume: basering is a Weyl algebra in characteristic 0

Purpose: check whether M is holonomic over the base ring

Note: M is holonomic if 2*dim(M) = dim(R), where R is the base ring; dim stands for Gelfand-Kirillov dimension

Example:

```
LIB "dmod.lib";
ring R = 0,(x,y),dp;
poly F = x*y*(x+y);
def A = annfsBM(F,0);
setring A;
LD;
↦ LD[1]=x*Dx+y*Dy+3
↦ LD[2]=x*y*Dy+y^2*Dy+x+2*y
↦ LD[3]=y^2*Dx*Dy-y^2*Dy^2+2*y*Dx-4*y*Dy-2
isHolonomic(LD);
↦ 1
ideal I = std(LD[1]);
I;
↦ I[1]=x*Dx+y*Dy+3
isHolonomic(I);
↦ 0
```

7.5.4.21 convloc

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: convloc(L); L a list

Return: list

Purpose: convert a ringlist L into another ringlist,
where all the 'p' orderings are replaced with the 's' orderings, e.g. `dp` by `ds`.

Assume: L is a result of a ringlist command

Example:

```
LIB "dmod.lib";
ring r = 0,(x,y,z),(Dp(2),dp(1));
list L = ringlist(r);
list N = convloc(L);
def rs = ring(N);
setring rs;
rs;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering Ds
↦ //                  : names    x y
↦ //        block   2 : ordering ds
↦ //                  : names    z
↦ //        block   3 : ordering C
```

7.5.4.22 minIntRoot

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: minIntRoot(P, fact); P an ideal, fact an int

Return: int

Purpose: minimal integer root of a maximal ideal P

Note: if fact==1, P is the result of some 'factorize' call,
else P is treated as the result of bernstein::gmssing.lib
in both cases without constants and multiplicities

Example:

```
LIB "dmod.lib";
ring r   = 0,(x,y),ds;
poly f1  = x*y*(x+y);
ideal I1 = bernstein(f1)[1]; // a local Bernstein poly
I1;
↦ I1[1]=-4/3
↦ I1[2]=-1
↦ I1[3]=-2/3
minIntRoot(I1,0);
↦ -1
poly  f2  = x2-y3;
ideal I2  = bernstein(f2)[1];
I2;
↦ I2[1]=-7/6
↦ I2[2]=-1
↦ I2[3]=-5/6
minIntRoot(I2,0);
↦ -1
// now we illustrate the behaviour of factorize
// together with a global ordering
ring r2  = 0,x,dp;
poly f3   = 9*(x+2/3)*(x+1)*(x+4/3); //global b-polynomial of f1=x*y*(x+y)
ideal I3 = factorize(f3,1);
I3;
↦ I3[1]=x+1
↦ I3[2]=3x+2
↦ I3[3]=3x+4
minIntRoot(I3,1);
↦ -1
// and a more interesting situation
ring  s  = 0,(x,y,z),ds;
poly  f  = x3 + y3 + z3;
ideal I  = bernstein(f)[1];
I;
↦ I[1]=-2
↦ I[2]=-5/3
↦ I[3]=-4/3
↦ I[4]=-1
minIntRoot(I,0);
↦ -2
```

7.5.4.23 isRational

Procedure from library `dmod.lib` (see Section 7.5.4 [dmod_lib], page 400).

Usage: isRational(n); n number

Return: int

Purpose: determine whether n is a rational number,
that is it does not contain parameters.

Assume: ground field is of characteristic 0

Example:

```
LIB "dmod.lib";
ring r = (0,a),(x,y),dp;
number n1 = 11/73;
isRational(n1);
↦ 1
number n2 = (11*a+3)/72;
isRational(n2);
↦ 0
```

7.5.5 dmodapp_lib

Library: dmodapp.lib

Purpose: Applications of algebraic D-modules

Authors: Viktor Levandovskyy, levandov@math.rwth-aachen.de
Daniel Andres, daniel.andres@math.rwth-aachen.de

Support: DFG Graduiertenkolleg 1632 'Experimentelle und konstruktive Algebra'

Overview: Let K be a field of characteristic 0, R = K[x1,...,xN] and D be the Weyl algebra in variables x1,...,xN,d1,...,dN. In this library there are the following procedures for algebraic D-modules:

- given a cyclic representation D/I of a holonomic module and a polynomial F in R, it is proved that the localization of D/I with respect to the mult. closed set of all powers of F is a holonomic D-module. Thus we aim to compute its cyclic representation D/L for an ideal L in D. The procedures for the localization are DLoc, SDLoc and DLoc0.

- annihilator in D of a given polynomial F from R as well as of a given rational function G/F from Quot(R). These can be computed via procedures annPoly resp. annRat.

- Groebner bases with respect to weights (according to (SST), given an arbitrary integer vector containing weights for variables, one computes the homogenization of a given ideal relative to this vector, then one computes a Groebner basis and returns the dehomogenization of the result), initial forms and initial ideals in Weyl algebras with respect to a given weight vector can be computed with GBWeight, inForm, initialMalgrange and initialIdealW.

- restriction and integration of a holonomic module D/I. Suppose I annihilates a function F(x1,...,xn). Our aim is to compute an ideal J directly from I, which annihilates

- F(0,...,0,xk,...,xn) in case of restriction or
- the integral of F with respect to x1,...,xm in case of integration. The corresponding procedures are restrictionModule, restrictionIdeal, integralModule and integralIdeal.

- characteristic varieties defined by ideals in Weyl algebras can be computed with charVariety and charInfo.

- appelF1, appelF2 and appelF4 return ideals in parametric Weyl algebras, which annihilate corresponding Appel hypergeometric functions.

References:

(SST) Saito, Sturmfels, Takayama 'Groebner Deformations of Hypergeometric Differential Equations', Springer, 2000
(OTW) Oaku, Takayama, Walther 'A Localization Algorithm for D-modules', Journal of Symbolic Computation, 2000
(OT) Oaku, Takayama 'Algorithms for D-modules',
Journal of Pure and Applied Algebra, 1998

Procedures: D-module See also: Section 7.5.2 [bfun_lib], page 375; Section 7.5.4 [dmod_lib], page 400; Section 7.5.7 [dmodvar_lib], page 453; Section D.6.13 [gmssing_lib], page 1702.

7.5.5.1 annPoly

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: annPoly(f); f a poly

Return: ring (a Weyl algebra) containing an ideal 'LD'

Purpose: compute the complete annihilator ideal of f in the corresponding
Weyl algebra

Assume: basering is commutative and over a field of characteristic 0

Note: Activate the output ring with the `setring` command.
In the output ring, the ideal 'LD' (in Groebner basis) is the
annihilator.

Display: If printlevel =1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z),dp;
poly f = x^2*z - y^3;
def A = annPoly(f);
setring A;    // A is the 3rd Weyl algebra in 6 variables
LD;           // the Groebner basis of annihilator
↦ LD[1]=Dz^2
↦ LD[2]=Dy*Dz
↦ LD[3]=Dx*Dy
↦ LD[4]=y*Dy+3*z*Dz-3
↦ LD[5]=x*Dx-2*z*Dz
↦ LD[6]=z*Dx*Dz-Dx
↦ LD[7]=Dy^3+3*Dx^2*Dz
↦ LD[8]=x*Dy^2+3*y*Dx*Dz
↦ LD[9]=x^2*Dy+3*y^2*Dz
↦ LD[10]=Dx^3
↦ LD[11]=3*y*Dx^2+z*Dy^2
↦ LD[12]=3*y^2*Dx+2*x*z*Dy
↦ LD[13]=y^3*Dz-x^2*z*Dz+x^2
gkdim(LD);    // must be 3 = 6/2, since A/LD is holonomic module
↦ 3
NF(Dy^4, LD); // must be 0 since Dy^4 clearly annihilates f
↦ 0
poly f = imap(r,f);
NF(LD*f,std(ideal(Dx,Dy,Dz))); // must be zero if LD indeed annihilates f
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
↦ _[6]=0
↦ _[7]=0
↦ _[8]=0
↦ _[9]=0
↦ _[10]=0
↦ _[11]=0
↦ _[12]=0
↦ _[13]=0
```

See also: Section 7.5.5.2 [annRat], page 422.

7.5.5.2 annRat

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: annRat(g,f); f, g polynomials

Return: ring (a Weyl algebra) containing an ideal 'LD'

Purpose: compute the annihilator of the rational function g/f in the corresponding Weyl algebra

Assume: basering is commutative and over a field of characteristic 0

Note: Activate the output ring with the `setring` command.
In the output ring, the ideal 'LD' (in Groebner basis) is the annihilator of g/f.
The algorithm uses the computation of Ann(f^{-1}) via D-modules, see (SST).

Display: If printlevel =1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),dp;
poly g = 2*x*y;  poly f = x^2 - y^3;
def B = annRat(g,f);
setring B;
LD;
↦ LD[1]=3*y^2*Dx^2*Dy+2*x*Dx*Dy^2+9*y*Dx^2+4*Dy^2
↦ LD[2]=3*y^3*Dx^2-10*x*y*Dx*Dy-8*y^2*Dy^2+10*x*Dx
↦ LD[3]=y^3*Dy^2-x^2*Dy^2-6*x*y*Dx+2*y^2*Dy+4*y
↦ LD[4]=3*x*Dx+2*y*Dy+1
↦ LD[5]=y^4*Dy-x^2*y*Dy+2*y^3+x^2
// Now, compare with the output of Macaulay2:
ideal tst = 3*x*Dx + 2*y*Dy + 1, y^3*Dy^2 - x^2*Dy^2 + 6*y^2*Dy + 6*y,
9*y^2*Dx^2*Dy-4*y*Dy^3+27*y*Dx^2+2*Dy^2, 9*y^3*Dx^2-4*y^2*Dy^2+10*y*Dy -10;
option(redSB); option(redTail);
LD = groebner(LD);
tst = groebner(tst);
print(matrix(NF(LD,tst)));  print(matrix(NF(tst,LD)));
↦ 0,0,0,0,0
↦ 0,0,0,0,0
// So, these two answers are the same
```

See also: Section 7.5.5.1 [annPoly], page 421.

7.5.5.3 DLoc

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: DLoc(I, f); I an ideal, f a poly

Return: list of ideal and list

Assume: the basering is a Weyl algebra

Purpose: compute the presentation of the localization of D/I w.r.t. f^s

Note: In the output list L,
- L[1] is an ideal (given as Groebner basis), the presentation of the localization,
- L[2] is a list containing roots with multiplicities of Bernstein polynomial of (D/I)_f.

Display: If printlevel =1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def R = Weyl();    setring R; // Weyl algebra in variables x,y,Dx,Dy
poly F = x2-y3;
ideal I = (y^3 - x^2)*Dx - 2*x, (y^3 - x^2)*Dy + 3*y^2; // I = Dx*F, Dy*F;
// I is not holonomic, since its dimension is not 4/2=2
gkdim(I);
↦ 3
list L = DLoc(I, x2-y3);
L[1]; // localized module (R/I)_f is isomorphic to R/LD0
↦ _[1]=3*x*Dx+2*y*Dy+12
↦ _[2]=3*y^2*Dx+2*x*Dy
↦ _[3]=y^3*Dy-x^2*Dy+6*y^2
L[2]; // description of b-function for localization
↦ [1]:
↦    _[1]=0
↦    _[2]=1/6
↦    _[3]=-1/6
↦ [2]:
↦    1,1,1
```

7.5.5.4 SDLoc

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: SDLoc(I, f); I an ideal, f a poly

Return: ring (basering extended by a new variable) containing an ideal 'LD'

Purpose: compute a generic presentation of the localization of D/I w.r.t. f^s

Assume: the basering D is a Weyl algebra over a field of characteristic 0

Note: Activate this ring with the `setring` command. In this ring,
the ideal LD (given as Groebner basis) is the presentation of the localization.

Display: If printlevel =1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def R = Weyl(); // Weyl algebra on the variables x,y,Dx,Dy
setring R;
poly F = x2-y3;
ideal I = Dx*F, Dy*F;
// note, that I is not holonomic, since it's dimension is not 2
gkdim(I);  // 3, while dim R = 4
↦ 3
def W = SDLoc(I,F);
setring W; // = R[s], where s is a new variable
LD;        // Groebner basis of s-parametric presentation
↦ LD[1]=3*x*Dx*s+2*y*Dy*s-6*s^2+6*s
↦ LD[2]=3*y^2*Dx*s+2*x*Dy*s
↦ LD[3]=y^3*Dy-x^2*Dy-3*y^2*s+3*y^2
↦ LD[4]=y^3*Dx-x^2*Dx+2*x*s-2*x
```

7.5.5.5 DLoc0

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: DLoc0(I, f); I an ideal, f a poly

Return: ring (a Weyl algebra) containing an ideal 'LD0' and a list 'BS'

Purpose: compute the presentation of the localization of D/I w.r.t. f^s,
where D is a Weyl Algebra, based on the output of procedure SDLoc

Assume: the basering is similar to the output ring of SDLoc procedure

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD0 (given as Groebner basis) is the presentation of the localization,
- the list BS contains roots and multiplicities of Bernstein polynomial of (D/I)_f.

Display: If printlevel =1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def R = Weyl();    setring R; // Weyl algebra in variables x,y,Dx,Dy
poly F = x2-y3;
ideal I = (y^3 - x^2)*Dx - 2*x, (y^3 - x^2)*Dy + 3*y^2; // I = Dx*F, Dy*F;
// moreover I is not holonomic, since its dimension is not 2 = 4/2
gkdim(I); // 3
↦ 3
def W = SDLoc(I,F);  setring W; // creates ideal LD in W = R[s]
def U = DLoc0(LD, x2-y3);  setring U; // compute in R
LD0; // Groebner basis of the presentation of localization
↦ LD0[1]=3*x*Dx+2*y*Dy+12
↦ LD0[2]=3*y^2*Dx+2*x*Dy
↦ LD0[3]=y^3*Dy-x^2*Dy+6*y^2
BS; // description of b-function for localization
↦ [1]:
↦    _[1]=0
↦    _[2]=1/6
↦    _[3]=-1/6
↦ [2]:
↦    1,1,1
```

7.5.5.6 GBWeight

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: GBWeight(I,u,v [,s,t,w]);
I ideal, u,v intvecs, s,t optional ints, w an optional intvec

Return: ideal, Groebner basis of I w.r.t. the weights u and v

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0
and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1
holds, i.e. the sequence of variables is given by
x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator
belonging to x(i).

Purpose: computes a Groebner basis with respect to given weights

Note: The weights u and v are understood as weight vectors for x(i) and D(i), respectively. According to (SST), one computes the homogenization of a given ideal relative to (u,v), then one computes a Groebner basis and returns the dehomogenization of the result.
If s<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
If t<>0, a matrix ordering is used for Groebner basis computations,
otherwise, and by default, a block ordering is used.
If w is given and consists of exactly 2*n strictly positive entries,
w is used for constructing the weighted homogenized Weyl algebra,
see Noro (2002). Otherwise, and by default, the homogenization weight (1,...,1) is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def D2 = Weyl();
setring D2;
ideal I = 3*x^2*Dy+2*y*Dx,2*x*Dx+3*y*Dy+6;
intvec u = -2,-3;
intvec v = -u;
GBWeight(I,u,v);
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
ideal J = std(I);
GBWeight(J,u,v); // same as above
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
u = 0,1;
GBWeight(I,u,v);
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=-x^3*Dx+y^2*Dx-3*x^2
```

7.5.5.7 initialMalgrange

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: initialMalgrange(f,[,a,b,v]); f poly, a,b optional ints, v opt. intvec

Return: ring, Weyl algebra induced by basering, extended by two new vars t,Dt

Purpose: computes the initial Malgrange ideal of a given polynomial w.r.t. the weight vector (-1,0...,0,1,0,...,0) such that the weight of t is -1
and the weight of Dt is 1.

Assume: The basering is commutative and over a field of characteristic 0.

Note: Activate the output ring with the `setring` command.
The returned ring contains the ideal 'inF', being the initial ideal
of the Malgrange ideal of f.
Varnames of the basering should not include t and Dt.
If a<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
If b<>0, a matrix ordering is used for Groebner basis computations,
otherwise, and by default, a block ordering is used.
If a positive weight vector v is given, the weight
(d,v[1],...,v[n],1,d+1-v[1],...,d+1-v[n]) is used for homogenization
computations, where d denotes the weighted degree of f.
Otherwise and by default, v is set to (1,...,1). See Noro (2002).

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),dp;
poly f = x^2+y^3+x*y^2;
def D = initialMalgrange(f);
setring D;
inF;
↦ inF[1]=x*Dt
↦ inF[2]=2*x*y*Dx+3*y^2*Dx-y^2*Dy-2*x*Dy
↦ inF[3]=2*x^2*Dx+x*y*Dx+x*y*Dy+18*t*Dt+9*x*Dx-x*Dy+6*y*Dy+4*x+18
↦ inF[4]=18*t*Dt^2+6*y*Dt*Dy-y*Dt+27*Dt
↦ inF[5]=y^2*Dt
↦ inF[6]=2*t*y*Dt+2*x*y*Dx+2*y^2*Dx-6*t*Dt-3*x*Dx-x*Dy-2*y*Dy+2*y-6
↦ inF[7]=x*y^2+y^3+x^2
↦ inF[8]=2*y^3*Dx-2*y^3*Dy-3*y^2*Dx-2*x*y*Dy+y^2*Dy-4*y^2+36*t*Dt+18*x*Dx+1\
   2*y*Dy+36
setring r;
intvec v = 3,2;
def D2 = initialMalgrange(f,1,1,v);
setring D2;
inF;
↦ inF[1]=x*Dt
↦ inF[2]=2*x*y*Dx+3*y^2*Dx-y^2*Dy-2*x*Dy
↦ inF[3]=4*x^2*Dx-3*y^2*Dx+2*x*y*Dy+y^2*Dy+36*t*Dt+18*x*Dx+12*y*Dy+8*x+36
↦ inF[4]=18*t*Dt^2+6*y*Dt*Dy-y*Dt+27*Dt
↦ inF[5]=y^2*Dt
↦ inF[6]=2*t*y*Dt-y^2*Dx+y^2*Dy-6*t*Dt-3*x*Dx+x*Dy-2*y*Dy+2*y-6
↦ inF[7]=x*y^2+y^3+x^2
↦ inF[8]=2*y^3*Dx-2*y^3*Dy-3*y^2*Dx-2*x*y*Dy+y^2*Dy-4*y^2+36*t*Dt+18*x*Dx+1\
   2*y*Dy+36
```

7.5.5.8 initialIdealW

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: initialIdealW(I,u,v [,s,t,w]);
I ideal, u,v intvecs, s,t optional ints, w an optional intvec

Return: ideal, GB of initial ideal of the input ideal wrt the weights u and v

Assume: The basering is the n-th Weyl algebra in characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Purpose: computes the initial ideal with respect to given weights.

Note: u and v are understood as weight vectors for x(1..n) and D(1..n) respectively.
If s<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
If t<>0, a matrix ordering is used for Groebner basis computations,
otherwise, and by default, a block ordering is used.
If w is given and consists of exactly 2*n strictly positive entries,
w is used as homogenization weight.
Otherwise, and by default, the homogenization weight (1,...,1) is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def D2 = Weyl();
setring D2;
ideal I = 3*x^2*Dy+2*y*Dx,2*x*Dx+3*y*Dy+6;
intvec u = -2,-3;
intvec v = -u;
initialIdealW(I,u,v);
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
ideal J = std(I);
initialIdealW(J,u,v); // same as above
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
u = 0,1;
initialIdealW(I,u,v);
↦ _[1]=Dx
↦ _[2]=Dy
```

7.5.5.9 inForm

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: inForm(I,w); I ideal or poly, w intvec

Return: ideal, generated by initial forms of generators of I w.r.t. w, or
poly, initial form of input poly w.r.t. w

Purpose: computes the initial form of an ideal or a poly w.r.t. the weight w

Note: The size of the weight vector must be equal to the number of variables of the basering.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def D = Weyl(); setring D;
poly F = 3*x^2*Dy+2*y*Dx;
poly G = 2*x*Dx+3*y*Dy+6;
ideal I = F,G;
intvec w1 = -1,-1,1,1;
intvec w2 = -1,-2,1,2;
intvec w3 = -2,-3,2,3;
inForm(I,w1);
↦ _[1]=2*y*Dx
↦ _[2]=2*x*Dx+3*y*Dy+6
inForm(I,w2);
```

```
↦ _[1]=3*x^2*Dy
↦ _[2]=2*x*Dx+3*y*Dy+6
inForm(I,w3);
↦ _[1]=3*x^2*Dy+2*y*Dx
↦ _[2]=2*x*Dx+3*y*Dy+6
inForm(F,w1);
↦ 2*y*Dx
```

7.5.5.10 restrictionIdeal

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: restrictionIdeal(I,w,[,eng,m,G]);
I ideal, w intvec, eng and m optional ints, G optional ideal

Return: ring (a Weyl algebra) containing an ideal 'resIdeal'

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by
x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that I is holonomic and that w is n-dimensional with non-negative entries.

Purpose: computes the restriction ideal of a holonomic ideal to the subspace defined by the variables corresponding to the non-zero entries of the given intvec

Note: The output ring is the Weyl algebra defined by the zero entries of w.
It contains an ideal 'resIdeal' being the restriction ideal of I wrt w.
If there are no zero entries, the input ring is returned.
If eng<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
The minimal integer root of the b-function of I wrt the weight (-w,w) can be specified via the optional argument m.
The optional argument G is used for specifying a Groebner basis of I wrt the weight (-w,w), that is, the initial form of G generates the initial ideal of I wrt the weight (-w,w).
Further note, that the assumptions on m and G (if given) are not checked.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(a,x,b,Da,Dx,Db),dp;
def D3 = Weyl();
setring D3;
ideal I = a*Db-Dx+2*Da,
x*Db-Da,
x*Da+a*Da+b*Db+1,
x*Dx-2*x*Da-a*Da,
b*Db^2+Dx*Da-Da^2+Db,
a*Dx*Da+2*x*Da^2+a*Da^2+b*Dx*Db+Dx+2*Da;
intvec w = 1,0,0;
def D2 = restrictionIdeal(I,w);
setring D2; D2;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x b Dx Db
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dbb=b*Db+1
resIdeal;
↦ resIdeal[1]=2*x*Db-Dx
↦ resIdeal[2]=x*Dx+2*b*Db+2
↦ resIdeal[3]=4*b*Db^2+Dx^2+6*Db
```

7.5.5.11 restrictionModule

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: restrictionModule(I,w,[,eng,m,G]);
I ideal, w intvec, eng and m optional ints, G optional ideal

Return: ring (a Weyl algebra) containing a module 'resMod'

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by
x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that I is holonomic and that w is n-dimensional with non-negative entries.

Purpose: computes the restriction module of a holonomic ideal to the subspace defined by the variables corresponding to the non-zero entries of the given intvec

Note: The output ring is the Weyl algebra defined by the zero entries of w.
It contains a module 'resMod' being the restriction module of I wrt w.
If there are no zero entries, the input ring is returned.
If eng<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
The minimal integer root of the b-function of I wrt the weight (-w,w) can be specified via the optional argument m.
The optional argument G is used for specifying a Groebner Basis of I wrt the weight (-w,w), that is, the initial form of G generates the initial ideal of I wrt the weight (-w,w).
Further note, that the assumptions on m and G (if given) are not checked.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(a,x,b,Da,Dx,Db),dp;
def D3 = Weyl();
setring D3;
ideal I = a*Db-Dx+2*Da, x*Db-Da, x*Da+a*Da+b*Db+1,
x*Dx-2*x*Da-a*Da, b*Db^2+Dx*Da-Da^2+Db,
a*Dx*Da+2*x*Da^2+a*Da^2+b*Dx*Db+Dx+2*Da;
intvec w = 1,0,0;
def rm = restrictionModule(I,w);
setring rm; rm;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x b Dx Db
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dbb=b*Db+1
print(resMod);
↦ 2*x*Db-Dx,x*Dx+2*b*Db+2,4*b*Db^2+Dx^2+6*Db
```

7.5.5.12 integralIdeal

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: integralIdeal(I,w,[,eng,m,G]);
I ideal, w intvec, eng and m optional ints, G optional ideal

Return: ring (a Weyl algebra) containing an ideal 'intIdeal'

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that I is holonomic and that w is n-dimensional with non-negative entries.

Purpose: computes the integral ideal of a holonomic ideal w.r.t. the subspace defined by the variables corresponding to the non-zero entries of the given intvec.

Note: The output ring is the Weyl algebra defined by the zero entries of w.
It contains ideal 'intIdeal' being the integral ideal of I w.r.t. w.
If there are no zero entries, the input ring is returned.
If eng<>0, std is used for Groebner basis computations, otherwise, and by default, slimgb is used.
The minimal integer root of the b-function of I wrt the weight (-w,w) can be specified via the optional argument m.
The optional argument G is used for specifying a Groebner basis of I wrt the weight (-w,w), that is, the initial form of G generates the initial ideal of I wrt the weight (-w,w).
Further note, that the assumptions on m and G (if given) are not checked.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,b,Dx,Db),dp;
def D2 = Weyl();
setring D2;
ideal I = x*Dx+2*b*Db+2, x^2*Dx+b*Dx+2*x;
intvec w = 1,0;
def D1 = integralIdeal(I,w);
setring D1; D1;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    b Db
↦ // noncommutative relations:
↦ //    Dbb=b*Db+1
intIdeal;
↦ intIdeal[1]=2*b*Db+1
```

7.5.5.13 integralModule

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: integralModule(I,w,[,eng,m,G]);
I ideal, w intvec, eng and m optional ints, G optional ideal

Return: ring (a Weyl algebra) containing a module 'intMod'

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that I is holonomic and that w is n-dimensional with non-negative entries.

Purpose: computes the integral module of a holonomic ideal w.r.t. the subspace defined by the variables corresponding to the non-zero entries of the given intvec

Note: The output ring is the Weyl algebra defined by the zero entries of w.
It contains a module 'intMod' being the integral module of I wrt w.
If there are no zero entries, the input ring is returned.
If eng<>0, `std` is used for Groebner basis computations, otherwise, and by default, `slimgb` is used.
Let F(I) denote the Fourier transform of I w.r.t. w.
The minimal integer root of the b-function of F(I) w.r.t. the weight (-w,w) can be specified via the optional argument m.
The optional argument G is used for specifying a Groebner Basis of F(I) wrt the weight (-w,w), that is, the initial form of G generates the initial ideal of F(I) w.r.t. the weight (-w,w).

Further note, that the assumptions on m and G (if given) are not checked.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,b,Dx,Db),dp;
def D2 = Weyl();
setring D2;
ideal I = x*Dx+2*b*Db+2, x^2*Dx+b*Dx+2*x;
intvec w = 1,0;
def im = integralModule(I,w);
setring im; im;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    b Db
↦ // noncommutative relations:
↦ //    Dbb=b*Db+1
print(intMod);
↦ 2*b*Db+1,0,
↦ 0,       b*Db
```

7.5.5.14 deRhamCohom

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: deRhamCohom(f[,w,eng,m]); f poly, w optional intvec,
eng and m optional ints

Return: ring (a Weyl Algebra) containing a list 'DR' of ideal and int

Assume: Basering is commutative and over a field of characteristic 0.

Purpose: computes a basis of the n-th de Rham cohomology group of the complement of the hypersurface defined by f, where n denotes the number of variables of the basering

Note: The output ring is the n-th Weyl algebra. It contains a list 'DR' with two entries (ideal J and int m) such that {f^m*J[i] : i=1..size(I)} is a basis of the n-th de Rham cohomology group of the complement of the hypersurface defined by f.
If w is an intvec with exactly n strictly positive entries, w is used in the computation. Otherwise, and by default, w is set to (1,...,1).
If eng<>0, `std` is used for Groebner basis computations,
otherwise, and by default, `slimgb` is used.
If m is given, it is assumed to be less than or equal to the minimal integer root of the Bernstein-Sato polynomial of f. This assumption is not checked. If not specified, m is set to the minimal integer root of the Bernstein-Sato polynomial of f.

Theory: (SST) pp. 232-235

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z),dp;
poly f = x^3+y^3+z^3;
def A = deRhamCohom(f); // we see that the K-dim is 2
setring A;
DR;
↦ [1]:
↦    _[1]=-x^3*Dx*Dy*Dz
↦    _[2]=-x*y*z*Dx*Dy*Dz
↦ [2]:
↦    -2
```

See also: Section 7.5.5.15 [deRhamCohomIdeal], page 434.

7.5.5.15 deRhamCohomIdeal

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: deRhamCohomIdeal (I[,w,eng,k,G]);
I ideal, w optional intvec, eng and k optional ints, G optional ideal

Return: ideal

Assume: The basering is the n-th Weyl algebra D over a field of characteristic zero and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by
x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that I is of special kind, namely let f in K[x] and consider the module K[x,1/f]f^m, where m is smaller than or equal to the minimal integer root of the Bernstein-Sato polynomial of f.
Since this module is known to be a holonomic D-module, it has a cyclic presentation D/I.

Purpose: computes a basis of the n-th de Rham cohomology group of the complement of the hypersurface defined by f

Note: The elements of the basis are of the form f^m*p, where p runs over the entries of the returned ideal.
If I does not satisfy the assumptions described above, the result might have no meaning. Note that I can be computed with `annfs`.
If w is an intvec with exactly n strictly positive entries, w is used in the computation. Otherwise, and by default, w is set to (1,...,1).
If eng<>0, `std` is used for Groebner basis computations, otherwise, and by default, `slimgb` is used.
Let F(I) denote the Fourier transform of I wrt w.
An integer smaller than or equal to the minimal integer root of the b-function of F(I) wrt the weight (-w,w) can be specified via the optional argument k.
The optional argument G is used for specifying a Groebner Basis of F(I) wrt the weight (-w,w), that is, the initial form of G generates the

initial ideal of F(I) wrt the weight (-w,w).
Further note, that the assumptions on I, k and G (if given) are not checked.

Theory: (SST) pp. 232-235

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z),dp;
poly F = x^3+y^3+z^3;
bfctAnn(F);            // Bernstein-Sato poly of F has minimal integer root -2
↦ [1]:
↦    _[1]=-1
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-2
↦ [2]:
↦    2,1,1,1
def W = annRat(1,F^2); // so we compute the annihilator of 1/F^2
setring W; W;          // Weyl algebra, contains LD = Ann(1/F^2)
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x y z Dx Dy Dz
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
↦ //    Dzz=z*Dz+1
LD;                    // K[x,y,z,1/F]F^(-2) is isomorphic to W/LD as W-module
↦ LD[1]=x*Dx+y*Dy+z*Dz+6
↦ LD[2]=z^2*Dy-y^2*Dz
↦ LD[3]=z^2*Dx-x^2*Dz
↦ LD[4]=y^2*Dx-x^2*Dy
↦ LD[5]=x^3*Dz+y^3*Dz+z^3*Dz+6*z^2
↦ LD[6]=x^3*Dy+y^3*Dy+y^2*z*Dz+6*y^2
deRhamCohomIdeal(LD);  // we see that the K-dim is 2
↦ _[1]=-x^3*Dx*Dy*Dz
↦ _[2]=-x*y*z*Dx*Dy*Dz
```

See also: Section 7.5.5.14 [deRhamCohom], page 433.

7.5.5.16 charVariety

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: charVariety(I [,eng]); I an ideal, eng an optional int

Return: ring (commutative) containing an ideal 'charVar'

Purpose: computes an ideal whose zero set is the characteristic variety of I in the sense of D-module theory

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Note: The output ring is commutative. It contains an ideal 'charVar'. If eng<>0, `std` is used for Groebner basis computations, otherwise, and by default, `slimgb` is used.

Display: If `printlevel`=1, progress debug messages will be printed, if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),Dp;
poly F = x3-y2;
printlevel = 0;
def A  = annfs(F);
setring A;      // Weyl algebra
LD;             // the annihilator of F
↦ LD[1]=2*x*Dx+3*y*Dy+6
↦ LD[2]=3*x^2*Dy+2*y*Dx
↦ LD[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ LD[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
def CA = charVariety(LD);
setring CA; CA; // commutative ring
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
charVar;
↦ charVar[1]=2*x*Dx+3*y*Dy
↦ charVar[2]=3*x^2*Dy+2*y*Dx
↦ charVar[3]=9*x*y*Dy^2-4*y*Dx^2
↦ charVar[4]=27*y^2*Dy^3+8*y*Dx^3
dim(std(charVar));   // hence I is holonomic
↦ 2
```

See also: Section 7.5.5.17 [charInfo], page 436.

7.5.5.17 charInfo

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: charInfo(I); I an ideal

Return: ring (commut.) containing ideals 'charVar','singLoc' and list 'primDec'

Purpose: computes characteristic variety of I (in the sense of D-module theory), its singular locus and primary decomposition

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Note: In the output ring, which is commutative:
- the ideal 'charVar' is the characteristic variety char(I),
- the ideal 'SingLoc' is the singular locus of char(I),
- the list 'primDec' is the primary decomposition of char(I).

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),Dp;
poly F = x3-y2;
printlevel = 0;
def A  = annfs(F);
setring A;      // Weyl algebra
LD;             // the annihilator of F
↦ LD[1]=2*x*Dx+3*y*Dy+6
↦ LD[2]=3*x^2*Dy+2*y*Dx
↦ LD[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ LD[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
def CA = charInfo(LD);
setring CA; CA; // commutative ring
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
charVar;        // characteristic variety
↦ charVar[1]=2*x*Dx+3*y*Dy
↦ charVar[2]=3*x^2*Dy+2*y*Dx
↦ charVar[3]=9*x*y*Dy^2-4*y*Dx^2
↦ charVar[4]=27*y^2*Dy^3+8*y*Dx^3
singLoc;        // singular locus
↦ singLoc[1]=y*Dy
↦ singLoc[2]=y*Dx
↦ singLoc[3]=2*x*Dx-3*y*Dy
↦ singLoc[4]=9*x*Dy^2-2*Dx^2
↦ singLoc[5]=3*x^2*Dy-y*Dx
↦ singLoc[6]=Dx^3
↦ singLoc[7]=x^3-y^2
primDec;        // primary decomposition
↦ [1]:
↦    [1]:
↦       _[1]=Dy
↦       _[2]=Dx
↦    [2]:
↦       _[1]=Dy
↦       _[2]=Dx
↦ [2]:
↦    [1]:
↦       _[1]=27*y*Dy^3+8*Dx^3
↦         _[2]=9*x*Dy^2-4*Dx^2
↦         _[3]=2*x*Dx+3*y*Dy
↦         _[4]=3*x^2*Dy+2*y*Dx
↦         _[5]=x^3-y^2
↦      [2]:
↦         _[1]=27*y*Dy^3+8*Dx^3
↦         _[2]=9*x*Dy^2-4*Dx^2
↦         _[3]=2*x*Dx+3*y*Dy
↦         _[4]=3*x^2*Dy+2*y*Dx
↦         _[5]=x^3-y^2
↦   [3]:
↦      [1]:
↦         _[1]=y
↦         _[2]=x
↦      [2]:
↦         _[1]=y
↦         _[2]=x
```

7.5.5.18 isFsat

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: isFsat(I, F); I an ideal, F a poly

Return: int, 1 if I is F-saturated and 0 otherwise

Purpose: checks whether the ideal I is F-saturated

Note: We check indeed that Ker(D–> F–> D/I) is 0, where D is the basering.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),dp;
poly G = x*(x-y)*y;
def A = annfs(G);
setring A;
poly F = x3-y2;
isFsat(LD,F);
↦ 1
ideal J = LD*F;
isFsat(J,F);
↦ 0
```

7.5.5.19 appelF1

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: appelF1();

Return: ring (a parametric Weyl algebra) containing an ideal 'IAppel1'

Purpose: defines the ideal in a parametric Weyl algebra,
which annihilates Appel F1 hypergeometric function

Note: The output ring is a parametric Weyl algebra. It contains an ideal
'IAappel1' annihilating Appel F1 hypergeometric function.
See (SST) p. 48.

Example:

```
LIB "dmodapp.lib";
def A = appelF1();
setring A;
IAppel1;
↦ IAppel1[1]=-x^3*Dx^2+x^2*Dx^2-x^2*y*Dx*Dy+x*y*Dx*Dy+(-a-b-1)*x^2*Dx+(c)*x\
   *Dx+(-b)*x*y*Dy+(-a*b)*x
↦ IAppel1[2]=-x*y^2*Dx*Dy+x*y*Dx*Dy-y^3*Dy^2+y^2*Dy^2+(-d)*x*y*Dx+(-a-d-1)*\
   y^2*Dy+(c)*y*Dy+(-a*d)*y
↦ IAppel1[3]=x*Dx*Dy-y*Dx*Dy+(-d)*Dx+(b)*Dy
```

7.5.5.20 appelF2

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: appelF2();

Return: ring (a parametric Weyl algebra) containing an ideal 'IAppel2'

Purpose: defines the ideal in a parametric Weyl algebra,
which annihilates Appel F2 hypergeometric function

Note: The output ring is a parametric Weyl algebra. It contains an ideal
'IAappel2' annihilating Appel F2 hypergeometric function.
See (SST) p. 85.

Example:

```
LIB "dmodapp.lib";
def A = appelF2();
setring A;
IAppel2;
↦ IAppel2[1]=-x^3*Dx^2+x^2*Dx^2-x^2*y*Dx*Dy+(-a-b-1)*x^2*Dx+x*Dx+(-b)*x*y*D\
   y+(-a*b)*x
↦ IAppel2[2]=-x*y^2*Dx*Dy-y^3*Dy^2+y^2*Dy^2+(-c)*x*y*Dx+(-a-c-1)*y^2*Dy+y*D\
   y+(-a*c)*y
```

7.5.5.21 appelF4

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: appelF4();

Return: ring (a parametric Weyl algebra) containing an ideal 'IAppel4'

Purpose: defines the ideal in a parametric Weyl algebra,
which annihilates Appel F4 hypergeometric function

Note: The output ring is a parametric Weyl algebra. It contains an ideal
'IAappel4' annihilating Appel F4 hypergeometric function.
See (SST) p. 39.

Example:

```
LIB "dmodapp.lib";
def A = appelF4();
setring A;
IAppel4;
↦ IAppel4[1]=-x^2*Dx^2+x*Dx^2-2*x*y*Dx*Dy-y^2*Dy^2+(-a-b-1)*x*Dx+(c)*Dx+(-a\
   -b-1)*y*Dy+(-a*b)
↦ IAppel4[2]=-x^2*Dx^2-2*x*y*Dx*Dy-y^2*Dy^2+y*Dy^2+(-a-b-1)*x*Dx+(-a-b-1)*y\
   *Dy+(d)*Dy+(-a*b)
```

7.5.5.22 fourier

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: fourier(I[,v]); I an ideal, v an optional intvec

Return: ideal

Purpose: computes the Fourier transform of an ideal in a Weyl algebra

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Note: The Fourier automorphism is defined by mapping x(i) to -D(i) and D(i) to x(i).
If v is an intvec with entries ranging from 1 to n, the Fourier transform of I restricted to the variables given by v is computed.

Example:

```
  LIB "dmodapp.lib";
  ring r = 0,(x,y,Dx,Dy),dp;
  def D2 = Weyl();
  setring D2;
  ideal I = x*Dx+2*y*Dy+2, x^2*Dx+y*Dx+2*x;
  intvec v = 2;
  fourier(I,v);
  ↦ _[1]=x*Dx-2*y*Dy
  ↦ _[2]=x^2*Dx-Dx*Dy+2*x
  fourier(I);
  ↦ _[1]=-x*Dx-2*y*Dy-1
  ↦ _[2]=x*Dx^2-x*Dy
```

See also: Section 7.5.5.23 [inverseFourier], page 440.

7.5.5.23 inverseFourier

Procedure from library dmodapp.lib (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: inverseFourier(I[,v]); I an ideal, v an optional intvec

Return: ideal

Purpose: computes the inverse Fourier transform of an ideal in a Weyl algebra

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Note: The Fourier automorphism is defined by mapping x(i) to -D(i) and D(i) to x(i).
If v is an intvec with entries ranging from 1 to n, the inverse Fourier transform of I restricted to the variables given by v is computed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def D2 = Weyl();
setring D2;
ideal I = x*Dx+2*y*Dy+2, x^2*Dx+y*Dx+2*x;
intvec v = 2;
ideal FI = fourier(I);
inverseFourier(FI);
↦ _[1]=x*Dx+2*y*Dy+2
↦ _[2]=x^2*Dx+y*Dx+2*x
```

See also: Section 7.5.5.22 [fourier], page 440.

7.5.5.24 bFactor

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: bFactor(f); f poly

Return: list of ideal and intvec and possibly a string

Purpose: tries to compute the roots of a univariate poly f

Note: The output list consists of two or three entries:
roots of f as an ideal, their multiplicities as intvec, and,
if present, a third one being the product of all irreducible factors of degree greater than one, given as string.
If f is the zero polynomial or if f has no roots in the ground field, this is encoded as root 0 with multiplicity 0.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),dp;
bFactor((x^2-1)^2);
↦ [1]:
↦    _[1]=1
↦    _[2]=-1
↦ [2]:
↦    2,2
bFactor((x^2+1)^2);
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    0
↦ [3]:
↦    x4+2x2+1
bFactor((y^2+1/2)*(y+9)*(y-7));
```

```
↦ [1]:
↦    _[1]=7
↦    _[2]=-9
↦ [2]:
↦    1,1
↦ [3]:
↦    2y2+1
bFactor(1);
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    0
↦ [3]:
↦    1
bFactor(0);
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    0
↦ [3]:
↦    0
```

7.5.5.25 intRoots

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: isInt(L); L a list

Return: list

Purpose: extracts integer roots from a list given in `bFactor` format

Assume: The input list must be given in the format of `bFactor`.

Note: Parameters are treated as integers.

Example:

```
LIB "dmodapp.lib";
ring r = 0,x,dp;
list L = bFactor((x-4/3)*(x+3)^2*(x-5)^4); L;
↦ [1]:
↦    _[1]=5
↦    _[2]=4/3
↦    _[3]=-3
↦ [2]:
↦    4,1,2
intRoots(L);
↦ [1]:
↦    _[1]=5
↦    _[2]=-3
↦ [2]:
↦    4,2
```

See also: Section 7.5.5.24 [bFactor], page 441.

7.5.5.26 poly2list

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: poly2list(f); f a poly

Return: list of exponents and corresponding terms of f

Purpose: converts a poly to a list of pairs consisting of intvecs (1st entry) and polys (2nd entry), where the i-th pair contains the exponent of the i-th term of f and the i-th term (with coefficient) itself.

Example:

```
LIB "dmodapp.lib";
ring r = 0,x,dp;
poly F = x;
poly2list(F);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       x
ring r2 = 0,(x,y),dp;
poly F = x2y+5xy2;
poly2list(F);
↦ [1]:
↦    [1]:
↦       2,1
↦    [2]:
↦       x2y
↦ [2]:
↦    [1]:
↦       1,2
↦    [2]:
↦       5xy2
poly2list(0);
↦ [1]:
↦    [1]:
↦       0,0
↦    [2]:
↦       0
```

7.5.5.27 fl2poly

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: fl2poly(L,s); L a list, s a string

Return: poly

Purpose: reconstruct a monic polynomial in one variable from its factorization

Assume: s is a string with the name of some variable and
L is supposed to consist of two entries:
- L[1] of the type ideal with the roots of a polynomial
- L[2] of the type intvec with the multiplicities of corr. roots

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z,s),Dp;
ideal I = -1,-4/3,0,-5/3,-2;
intvec mI = 2,1,2,1,1;
list BS = I,mI;
poly p = fl2poly(BS,"s");
p;
↦ s7+7s6+173/9s5+233/9s4+154/9s3+40/9s2
factorize(p,2);
↦ [1]:
↦    _[1]=s+2
↦    _[2]=3s+4
↦    _[3]=3s+5
↦    _[4]=s
↦    _[5]=s+1
↦ [2]:
↦    1,1,1,2,2
```

7.5.5.28 insertGenerator

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: insertGenerator(id,p[,k]);
id an ideal/module, p a poly/vector, k an optional int

Return: of the same type as id

Purpose: inserts p into id at k-th position and returns the enlarged object

Note: If k is given, p is inserted at position k, otherwise (and by default), p is inserted at the beginning (k=1).

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z),dp;
ideal I = x^2,z^4;
insertGenerator(I,y^3);
↦ _[1]=y3
↦ _[2]=x2
↦ _[3]=z4
insertGenerator(I,y^3,2);
↦ _[1]=x2
↦ _[2]=y3
↦ _[3]=z4
module M = I*gen(3);
insertGenerator(M,[x^3,y^2,z],2);
↦ _[1]=x2*gen(3)
↦ _[2]=x3*gen(1)+y2*gen(2)+z*gen(3)
↦ _[3]=z4*gen(3)
insertGenerator(M,x+y+z,4);
↦ _[1]=x2*gen(3)
↦ _[2]=z4*gen(3)
↦ _[3]=0
↦ _[4]=x*gen(1)+y*gen(1)+z*gen(1)
```

See also: Section 7.5.5.29 [deleteGenerator], page 445.

7.5.5.29 deleteGenerator

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: deleteGenerator(id,k); id an ideal/module, k an int

Return: of the same type as id

Purpose: deletes the k-th generator from the first argument and returns the altered object

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y,z),dp;
ideal I = x^2,y^3,z^4;
deleteGenerator(I,2);
↦ _[1]=x2
↦ _[2]=z4
module M = [x,y,z],[x2,y2,z2],[x3,y3,z3];
print(deleteGenerator(M,2));
↦ x,x3,
↦ y,y3,
↦ z,z3
M = M[1];
deleteGenerator(M,1);
↦ _[1]=0
```

See also: Section 7.5.5.28 [insertGenerator], page 444.

7.5.5.30 isInt

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: isInt(n); n a number

Return: int, 1 if n is an integer or 0 otherwise

Purpose: check whether given object of type number is actually an int

Note: Parameters are treated as integers.

Example:

```
LIB "dmodapp.lib";
ring r = 0,x,dp;
number n = 4/3;
isInt(n);
↦ 0
n = 11;
isInt(n);
↦ 1
```

7.5.5.31 sortIntvec

Procedure from library `dmodapp.lib` (see Section 7.5.5 [dmodapp_lib], page 420).

Usage: sortIntvec(v); v an intvec

Return: list of two intvecs

Purpose: sorts an intvec

Note: In the output list L, the first entry consists of the entries of v satisfying L[1][i] >= L[1][i+1]. The second entry is a permutation such that v[L[2]] = L[1].
Unlike in the procedure `sort`, zeros are not dismissed.

Example:

```
LIB "dmodapp.lib";
ring r = 0,x,dp;
intvec v = -1,0,1,-2,0,2;
list L = sortIntvec(v); L;
↦ [1]:
↦    2,1,0,0,-1,-2
↦ [2]:
↦    6,3,2,5,1,4
v[L[2]];
↦ 2 1 0 0 -1 -2
v = -3,0;
sortIntvec(v);
↦ [1]:
↦    0,-3
↦ [2]:
↦    2,1
v = 0,-3;
sortIntvec(v);
↦ [1]:
↦    0,-3
↦ [2]:
↦    1,2
```

See also: Section D.2.3.15 [sort], page 812.

7.5.6 dmodideal_lib

Library: dmodideal.lib

Purpose: Algorithms for Bernstein-Sato ideals of morphisms

Authors: Robert Loew, robert.loew at rwth-aachen.de
Viktor Levandovskyy, levandov at math.rwth-aachen.de Jorge Martin Morales, jorge at unizar.es

Overview: Let K be a field of characteristic 0. Given a polynomial ring R = K[x_1,...,x_n] and a map, given by polynomials F_1,...,F_r from R, one is interested in the R[1/(F_1*...*F_r)]-module of rank one, generated by the symbol F^s=F_1^(s_1)*...*F_r^(s_r) for symbolic discrete variables s_1,...,s_r. This module R[1/(F_1*...*F_r)]*F^s has a structure of a D(R)[s_1,...,s_r]-module, where D(R) is an n-th Weyl algebra K<x_1,...,x_n,d_1,...,d_n | d_j x_j = x_j d_j +1> and D(R)[s] := D(R) tensored with K[s]:=K[s_1,...,s_r] over K. We often write just D for D(R) and D[s] for D(R)[s].

One is interested in the computation of the following data:
- Ann_{D[s]} F^s, the annihilator of F^s in D[s]; see annihilatorMultiFs
- Ann^{1}_{D[s]} F^s, the logarithmic annihilator of F^s in D[s]; see annfsLogIdeal
- several kinds of global Bernstein-Sato ideals in K[s], cf. (CU) and (Bud12); see BernsteinSatoIdeal and BSidealFromAnn

- Ann_{D} F^alpha for alpha from K^r, the annihilator of F^alpha in D; see annfalphaI
- sub- and over-ideals, bounding the Bernstein-Sato ideal; see BFBoundsBudur

References:

(BM) the Ann F^s algorithm by Briancon and Maisonobe (Remarques sur l'ideal de Bernstein associe a des polynomes, preprint, 2002)
(LM08) V. Levandovskyy and J. Martin-Morales, ISSAC 2008
(CU) Castro and Ucha, On the computation of Bernstein-Sato ideals, JSC 2005
(SST) Saito, Sturmfels, Takayama 'Groebner Deformations of Hypergeometric Differential Equations', Springer, 2000
(Bud12) N. Budur, Bernstein-Sato ideals and local systems, Annales de l'Institut Fourier, Volume 65 (2015) no. 2
(OT99) T. Oaku and N. Takayama, An algorithm for de Rham cohomology groups of the complement of an affine variety via D-module computation, Journal of Pure and Applied Algebra, 1999

Procedures: See also: Section 7.5.2 [bfun_lib], page 375; Section 7.5.4 [dmod_lib], page 400; Section 7.5.5 [dmodapp_lib], page 420; Section 7.5.14 [dmodloc_lib], page 522; Section D.6.13 [gmssing_lib], page 1702.

7.5.6.1 annfsLogIdeal

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: annfsLogIdeal(F); F an ideal

Return: ring

Purpose: compute the logarithmic annihilator of F[1]^s(1)*...*F[P]^s(P)

Assume: basering is a commutative polynomial ring in characteristic 0

Note: activate the output ring with the `setring` command. In this ring, annfsLog is the logarithmic annihilator of F^s (no Groebner basis). If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y),dp;
ideal F = x^3+y^4+x*y^3, x;
def S1 = annfsLogIdeal(F);
setring S1;
annfsLog;
↦ annfsLog[1]=-9*s(1)*x-12*s(1)*y-3*s(2)*x-4*s(2)*y+3*x^2*Dx+4*x*y*Dx+2*x*y\
   *Dy+3*y^2*Dy
↦ annfsLog[2]=-36*s(1)*y^2-192*s(1)*y-9*s(2)*y^2-64*s(2)*y+9*x*y^2*Dx+9*y^3\
   *Dy+64*x*y*Dx+3*x^2*Dy-4*x*y*Dy+48*y^2*Dy
setring R; // now compare with the full annihilator
def S = annihilatorMultiFs(F);
setring S;
annFs;
↦ annFs[1]=-3*x^2*Dx-4*x*y*Dx-2*x*y*Dy-3*y^2*Dy+9*x*s(1)+12*y*s(1)+3*x*s(2)\
   +4*y*s(2)
↦ annFs[2]=-9*x*y^2*Dx-9*y^3*Dy-64*x*y*Dx-3*x^2*Dy+4*x*y*Dy-48*y^2*Dy+36*y^\
   2*s(1)+9*y^2*s(2)+192*y*s(1)+64*y*s(2)
↦ annFs[3]=-3*x^2*y^2*Dx-4*x*y^3*Dx+x*y^3*Dy+3*x^3*Dy+3*x*y^2*s(2)+4*y^3*s(\
   2)
↦ annFs[4]=-144*x^2*y*Dx^2-84*x*y^2*Dx^2-141*x*y^2*Dx*Dy-81*y^3*Dy^2-768*x^\
   2*Dx^2-256*x*y*Dx^2+84*x^2*Dx*Dy-1008*x*y*Dx*Dy-192*y^2*Dx*Dy-51*x^2*Dy^2\
   +64*x*y*Dy^2-336*y^2*Dy^2+36*x*y*Dx*s(2)+84*y^2*Dx*s(2)-27*y^2*Dy*s(2)-70\
   2*x*y*Dx-84*y^2*Dx-555*y^2*Dy+768*y*Dx*s(1)-384*y*Dy*s(1)+1296*y*s(1)^2+2\
   56*y*Dx*s(2)-128*y*Dy*s(2)+756*y*s(1)*s(2)+108*y*s(2)^2-3712*x*Dx-256*y*D\
   x+244*x*Dy-2528*y*Dy+1980*y*s(1)+6912*s(1)^2+558*y*s(2)+4608*s(1)*s(2)+76\
   8*s(2)^2+8832*s(1)+2944*s(2)
lead(groebner(imap(S1,annfsLog)));
↦ _[1]=3*x^2*Dx
↦ _[2]=9*x*y^2*Dx
↦ _[3]=x*y^3*Dy
↦ _[4]=27*y^4*Dx*Dy
lead(groebner(annFs)); // and we see the difference
↦ _[1]=3*x^2*Dx
↦ _[2]=9*x*y^2*Dx
↦ _[3]=3*y^3*Dx*Dy
↦ _[4]=x*y^3*Dy
```

7.5.6.2 annihilatorMultiFs

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: annihilatorMultiFs(F [,eng,us,ord]); F an ideal, eng, us, ord optional ints

Return: ring

Purpose: compute Ann(F[1]^s(1)*...*F[P]^s(P))
with the multivariate algorithm by Briancon and Maisonobe.

Assume: basering is a commutative polynomial ring in characteristic 0

Note: activate the output ring with the `setring` command. In this ring, the ideal annFs is the annihilator of F[1]^s_1*..*F[P]^s_p. If eng <>0, `std` is used for Groebner basis computations, otherwise, and by default `slimgb` is used. If us<>0, then syzygies-driven method is used additionally. If specified, ord describes the desired order from the following choices: 0 - 'dp'
1 - elimination order for x, 'dp' in the parts
2 - elimination order for s, 'dp' in the parts
3 - elimination order for x and s, 'dp' in the parts
4 - elimination order for x and D, 'dp' in the parts
(used for the further work in the Bernstein-Sato ideal) If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y),dp;
ideal F = x^2-y,y;
def S = annihilatorMultiFs(F,0,0,0);
setring S;
annFs;
↦ annFs[1]=-2*x*y*Dy-y*Dx+2*x*s(2)
↦ annFs[2]=-x*Dx-2*y*Dy+2*s(1)+2*s(2)
groebner(annFs);
↦ _[1]=x*Dx+2*y*Dy-2*s(1)-2*s(2)
↦ _[2]=2*x*y*Dy+y*Dx-2*x*s(2)
↦ _[3]=4*y^2*Dy^2-y*Dx^2-4*y*Dy*s(1)-8*y*Dy*s(2)+2*y*Dy+4*s(1)*s(2)+4*s(2)^\
   2+2*s(2)
```

7.5.6.3 BSidealFromAnn

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: BSidealFromAnn(F, @R [,eng,met]); F an ideal, @R a ring, eng, met optional ints

Return: ring

Purpose: compute several kinds of Bernstein-Sato ideals, associated to f = F[1]*..*F[P], with the multivariate algorithm by Briancon and Maisonobe from ann(F^s) as input.

Assume: basering is a commutative polynomial ring in characteristic 0 @R is a ring as returned from annihilatorMultiFs.

Note: activate the output ring with the `setring` command. In this ring, the ideal BS is a Bernstein-Sato ideal of a polynomial f = F[1]*..*F[P]. If eng <>0, `std` is used for Groebner basis computations, otherwise, and by default `slimgb` is used. If met is of type int:
if met <0, the B-Sigma ideal (cf. (CU)) is computed.
If 0 < met < P, then the ideal B_met (cf. (CU)) is computed. If met is an intvec or a list of intvecs, Budurs generalized Bernstein-Sato ideal associated to met is computed. Otherwise, and by default, the ideal B (cf. (CU)) is computed. If met is of type intvec: Budurs generalized Bernstein-Sato ideal B^met_F is computed. If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y),dp;
ideal F = x+y,x-y,x;
def @R = annihilatorMultiFs(F, 0, 0, 4);
// first we compute the ideal B
def @R2 = BSidealFromAnn(F, @R, 0, 0);
setring @R2;
BS;
↦ BS[1]=s(1)^4*s(2)*s(3)+s(1)^4*s(2)+s(1)^4*s(3)+s(1)^4+3*s(1)^3*s(2)^2*s(3\
   )+3*s(1)^3*s(2)^2+3*s(1)^3*s(2)*s(3)^2+16*s(1)^3*s(2)*s(3)+13*s(1)^3*s(2)\
   +3*s(1)^3*s(3)^2+13*s(1)^3*s(3)+10*s(1)^3+3*s(1)^2*s(2)^3*s(3)+3*s(1)^2*s\
   (2)^3+6*s(1)^2*s(2)^2*s(3)^2+30*s(1)^2*s(2)^2*s(3)+24*s(1)^2*s(2)^2+3*s(1\
   )^2*s(2)*s(3)^3+30*s(1)^2*s(2)*s(3)^2+83*s(1)^2*s(2)*s(3)+56*s(1)^2*s(2)+\
   3*s(1)^2*s(3)^3+24*s(1)^2*s(3)^2+56*s(1)^2*s(3)+35*s(1)^2+s(1)*s(2)^4*s(3\
   )+s(1)*s(2)^4+3*s(1)*s(2)^3*s(3)^2+16*s(1)*s(2)^3*s(3)+13*s(1)*s(2)^3+3*s\
   (1)*s(2)^2*s(3)^3+30*s(1)*s(2)^2*s(3)^2+83*s(1)*s(2)^2*s(3)+56*s(1)*s(2)^\
   2+s(1)*s(2)*s(3)^4+16*s(1)*s(2)*s(3)^3+83*s(1)*s(2)*s(3)^2+162*s(1)*s(2)*\
   s(3)+94*s(1)*s(2)+s(1)*s(3)^4+13*s(1)*s(3)^3+56*s(1)*s(3)^2+94*s(1)*s(3)+\
   50*s(1)+s(2)^4*s(3)+s(2)^4+3*s(2)^3*s(3)^2+13*s(2)^3*s(3)+10*s(2)^3+3*s(2\
   )^2*s(3)^3+24*s(2)^2*s(3)^2+56*s(2)^2*s(3)+35*s(2)^2+s(2)*s(3)^4+13*s(2)*\
   s(3)^3+56*s(2)*s(3)^2+94*s(2)*s(3)+50*s(2)+s(3)^4+10*s(3)^3+35*s(3)^2+50*\
   s(3)+24
setring R;
// secondly we compute the ideal B_1
```

```
@R2 = BSidealFromAnn(F, @R, 0, 1);
setring @R2;
BS;
↦ BS[1]=s(1)^2+s(1)*s(2)+s(1)*s(3)+3*s(1)+s(2)+s(3)+2
```

7.5.6.4 BernsteinSatoIdeal

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: BernsteinSatoIdeal(F [,eng,met,us]); F an ideal, eng, us optional ints, met optional int or intvec

Return: ring

Purpose: compute two kinds of Bernstein-Sato ideals, associated to f = F[1]*..*F[P], with the multivariate algorithm by Briancon and Maisonobe.

Assume: basering is a commutative polynomial ring in characteristic 0

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD is the annihilator of F[1]^s_1*..*F[P]^s_p,
- the list or ideal BS is a Bernstein-Sato ideal of a polynomial f = F[1]*..*F[P]. If eng <>0, `std` is used for Groebner basis computations, otherwise, and by default `slimgb` is used. If met <0, the B-Sigma ideal (cf. Castro and Ucha, 'On the computation of Bernstein-Sato ideals', 2005) is computed. If 0 < met < P, then the ideal B_P (cf. the paper) is computed. If met is an intvec, Budurs generalized Bernstein-Sato ideal associated to met is computed.
Otherwise, and by default, the ideal B (cf. the paper) is computed. If us<>0, then syzygies-driven method is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y),dp;
ideal F = x^2-y,y;
// first we compute the ideal B:
def S = BernsteinSatoIdeal(F);
setring S;
BS;
↦ BS[1]=4*s(1)^3*s(2)+4*s(1)^3+8*s(1)^2*s(2)^2+28*s(1)^2*s(2)+20*s(1)^2+4*s\
   (1)*s(2)^3+28*s(1)*s(2)^2+55*s(1)*s(2)+31*s(1)+4*s(2)^3+20*s(2)^2+31*s(2)\
   +15
// secondly we compute the ideal B_1:
setring R;
def S = BernsteinSatoIdeal(F,0,1);
↦ // ** redefining S (def S = BernsteinSatoIdeal(F,0,1);) ./examples/Bernst\
   einSatoIdeal.sing:10
setring S;
BS;
↦ BS[1]=2*s(1)^2+2*s(1)*s(2)+5*s(1)+2*s(2)+3
// thirdly we compute the ideal B_sigma:
setring R;
def S = BernsteinSatoIdeal(F,0,-1);
↦ // ** redefining S (def S = BernsteinSatoIdeal(F,0,-1);) ./examples/Berns\
   teinSatoIdeal.sing:15
setring S;
BS;
↦ BS[1]=2*s(1)*s(2)+2*s(1)+2*s(2)^2+5*s(2)+3
↦ BS[2]=2*s(1)^2+3*s(1)-2*s(2)^2-3*s(2)
```

7.5.6.5 BFBoundsBudur

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: BFBoundsBudur(F,m); F an ideal, m an intvec

Return: ring

Assume: basering is a commutative polynomial ring in characteristic 0

Purpose: determine upper and lower bounds of the Bernstein-Sato ideal associated to m with the method of (Bud12)

Note: The returned ring contains lists Bj, containing the Bernstein-Sato ideals associated to e_j,
shiftedIdeals, containing the shifted ideals from (Bud12) 4.7, and ideals upperBound, lowerBound which give upper bound and lower bound for the Bernstein-Sato-Ideal associated to m respectively.

Example:

```
  LIB "dmodideal.lib";
  ring r = 0,(x,y,z),dp;
  setring r;
  ideal F = x*z,2*x^2*y^2*z+x^4+y^4;
  def A = BFBoundsBudur(F,intvec(1,1));
  setring A;
  lead(upperBound);
  ↦ _[1]=2*s(1)^8*s(2)^2
  ↦ _[2]=s(1)^9*s(2)
  lead(lowerBound);
  ↦ _[1]=s(1)^11*s(2)
  ↦ _[2]=2*s(1)^10*s(2)^2
  ↦ _[3]=2*s(1)^10*s(2)^2
  ↦ _[4]=4*s(1)^9*s(2)^3
```

7.5.6.6 annfalphaI

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: annfalphaI(f,alpha); f,alpha ideals

Return: ring

Assume: basering is a commutative polynomial ring in characteristic 0

Purpose: determine annihilator of f^alpha with the method of (OT99)

Note: The returned ring contains the annihilator of f^alpha over D as annfalpha. alpha should contain the desired rational exponents.
The procedure may also be applied to the univariate case, i.e. for r=1.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y,z),dp;
ideal f = x,y,z;
ideal alpha = 1/4, 2/3, 1;
def A = annfalphaI(f,alpha);
setring A;
annfalpha;
↦ annfalpha[1]=Dz^2
↦ annfalpha[2]=z*Dz-1
↦ annfalpha[3]=3*y*Dy-2
↦ annfalpha[4]=4*x*Dx-1
```

7.5.6.7 extractS

Procedure from library `dmodideal.lib` (see Section 7.5.6 [dmodideal_lib], page 446).

Usage: extractS(I,r); I ideal, r int

Return: ring

Assume: I is an ideal in the first r variables of the basering and these r variables generate a commutative subring

Purpose: give the ideal generated by I in the commutative subring generated by the first r variables, ordering dp

Note: The returned ring contains I.

Example:

```
LIB "dmodideal.lib";
ring R = 0,(x,y),dp;
ideal f = x^2-y^2,y;
def S = BernsteinSatoIdeal(f);
setring S;
BS;
↦ BS[1]=8*s(1)^4*s(2)+8*s(1)^4+12*s(1)^3*s(2)^2+56*s(1)^3*s(2)+44*s(1)^3+6*\
   s(1)^2*s(2)^3+54*s(1)^2*s(2)^2+136*s(1)^2*s(2)+88*s(1)^2+s(1)*s(2)^4+16*s\
   (1)*s(2)^3+77*s(1)*s(2)^2+138*s(1)*s(2)+76*s(1)+s(2)^4+10*s(2)^3+35*s(2)^\
   2+50*s(2)+24
def T = extractS(BS,2);
setring T;
I;
↦ I[1]=8*s(1)^4*s(2)+12*s(1)^3*s(2)^2+6*s(1)^2*s(2)^3+s(1)*s(2)^4+8*s(1)^4+\
   56*s(1)^3*s(2)+54*s(1)^2*s(2)^2+16*s(1)*s(2)^3+s(2)^4+44*s(1)^3+136*s(1)^\
   2*s(2)+77*s(1)*s(2)^2+10*s(2)^3+88*s(1)^2+138*s(1)*s(2)+35*s(2)^2+76*s(1)\
   +50*s(2)+24
factorize(I[1]);
↦ [1]:
↦    _[1]=1
↦    _[2]=s(1)+1
↦    _[3]=s(2)+1
↦    _[4]=2*s(1)+s(2)+2
↦    _[5]=2*s(1)+s(2)+3
↦    _[6]=2*s(1)+s(2)+4
↦ [2]:
↦    1,1,1,1,1,1
```

7.5.7 dmodvar_lib

Library: dmodvar.lib

Purpose: Algebraic D-modules for varieties

Authors: Daniel Andres, daniel.andres@math.rwth-aachen.de
Viktor Levandovskyy, levandov@math.rwth-aachen.de
Jorge Martin-Morales, jorge@unizar.es

Support: DFG Graduiertenkolleg 1632 'Experimentelle und konstruktive Algebra'

Overview: Let K be a field of characteristic 0. Given a polynomial ring R = K[x_1,...,x_n] and polynomials f_1,...,f_r in R, define F = f_1*...*f_r and F^s = f_1^s_1*...*f_r^s_r for symbolic discrete (that is shiftable) variables s_1,..., s_r. The module R[1/F]*F^s has the structure of a D<S>-module, where D<S> = D(R) tensored with S over K, where
- D(R) is the n-th Weyl algebra K<x_1,...,x_n,d_1,...,d_n | d_j x_j = x_j d_j + 1>
- S is the universal enveloping algebra of gl_r, generated by s_i = s_{ii}.
One is interested in the following data:
- the left ideal Ann F^s in D<S>, usually denoted by LD in the output
- global Bernstein polynomial in one variable s = s_1+...+s_r, denoted by bs,
- its minimal integer root s0, the list of all roots of bs, which are known to be negative rational numbers, with their multiplicities, which is denoted by BS
- an r-tuple of operators in D<S>, denoted by PS, such that the functional equality sum(k=1 to k=r) P_k*f_k*F^s = bs*F^s holds in R[1/F]*F^s.

References:

(BMS06) Budur, Mustata, Saito: Bernstein-Sato polynomials of arbitrary varieties (2006).
(ALM09) Andres, Levandovskyy, Martin-Morales: Principal Intersection and Bernstein-Sato Polynomial of an Affine Variety (2009).

Procedures: See also: Section 7.5.2 [bfun_lib], page 375; Section 7.5.4 [dmod_lib], page 400; Section 7.5.5 [dmodapp_lib], page 420; Section D.6.13 [gmssing_lib], page 1702.

7.5.7.1 bfctVarIn

Procedure from library `dmodvar.lib` (see Section 7.5.7 [dmodvar_lib], page 453).

Usage: bfctVarIn(I [,a,b,c]); I an ideal, a,b,c optional ints

Return: list of ideal and intvec

Purpose: computes the roots of the Bernstein-Sato polynomial and their multiplicities for an affine algebraic variety defined by I.

Assume: The basering is commutative and over a field of characteristic 0.
Varnames of the basering do not include t(1),...,t(r) and Dt(1),...,Dt(r), where r is the number of entries of the input ideal.

Note: In the output list, say L,
- L[1] of type ideal contains all the rational roots of a b-function,
- L[2] of type intvec contains the multiplicities of above roots,
- optional L[3] of type string is the part of b-function without rational roots.
Note, that a b-function of degree 0 is encoded via L[1][1]=0, L[2]=0 and L[3] is 1 (for nonzero constant) or 0 (for zero b-function).
If a<>0, the ideal is used as given. Otherwise, and by default, a heuristically better

suited generating set is used to reduce computation time.
If b<>0, `std` is used for GB computations in characteristic 0, otherwise, and by default, `slimgb` is used.
If c<>0, a matrix ordering is used for GB computations, otherwise, and by default, a block ordering is used.
Further note, that in this proc, the initial ideal of the multivariate Malgrange ideal defined by I is computed and then a system of linear equations is solved by linear reductions following the ideas by Noro.
The result is shifted by 1-codim(Var(F)) following (BMS06).

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodvar.lib";
ring R = 0,(x,y,z),dp;
ideal F = x^2+y^3, z;
list L = bfctVarIn(F);
L;
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
```

7.5.7.2 bfctVarAnn

Procedure from library `dmodvar.lib` (see Section 7.5.7 [dmodvar_lib], page 453).

Usage: bfctVarAnn(F[,gid,eng]); F an ideal, gid,eng optional ints

Return: list of an ideal and an intvec

Purpose: computes the roots of the Bernstein-Sato polynomial and their multiplicities for an affine algebraic variety defined by F = F[1],..,F[r].

Assume: The basering is commutative and over a field in char 0.

Note: In the output list, the ideal contains all the roots and the intvec their multiplicities.
If gid<>0, the ideal is used as given. Otherwise, and by default, a heuristically better suited generating set is used.
If eng<>0, `std` is used for GB computations, otherwise, and by default, `slimgb` is used.
Computational remark: The time of computation can be very different depending on the chosen generators of F, although the result is always the same.
Further note that in this proc, the annihilator of f^s in D[s] is computed and then a system of linear equations is solved by linear reductions in order to find the minimal polynomial of S = s(1)(1) + ... + s(P)(P). The resulted is shifted by 1-codim(Var(F)) following (BMS06).

Display: If printlevel=1, progress debug messages will be printed,
if printlevel=2, all the debug messages will be printed.

Example:

```
LIB "dmodvar.lib";
ring R = 0,(x,y,z),Dp;
```

```
ideal F = x^2+y^3, z;
bfctVarAnn(F);
↦ [1]:
↦    _[1]=-5/6
↦    _[2]=-1
↦    _[3]=-7/6
↦ [2]:
↦    1,1,1
```

7.5.7.3 SannfsVar

Procedure from library `dmodvar.lib` (see Section 7.5.7 [dmodvar_lib], page 453).

Usage: SannfsVar(F [,ORD,eng]); F an ideal, ORD an optional string, eng an optional int

Return: ring (Weyl algebra tensored with U(gl_P)), containing an ideal LD

Purpose: compute the D<S>-module structure of D<S>*f^s where f = F[1]*...*F[P] and D<S> is the Weyl algebra D tensored with K<S>=U(gl_P), according to the generalized algorithm by Briancon and Maisonobe for affine varieties

Assume: The basering is commutative and over a field of characteristic 0.

Note: Activate the output ring D<S> with the `setring` command. In the ring D<S>, the ideal LD is the needed D<S>-module structure.
The value of ORD must be an elimination ordering in D<Dt,S> for Dt written in the string form, otherwise the result may have no meaning. By default ORD = '(a(1..(P)..1),a(1..(P+P^2)..1),dp)'.
If eng<>0, `std` is used for Groebner basis computations, otherwise, and by default `slimgb` is used.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodvar.lib";
ring R = 0,(x,y),Dp;
ideal F = x^3, y^5;
//ORD = "(a(1,1),a(1,1,1,1,1,1),dp)";
//eng = 0;
def A = SannfsVar(F);
setring A;
A;
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering a
↦ //                  : names    s(1)(1) s(1)(2) s(2)(1) s(2)(2)
↦ //                  : weights        1       1       1       1
↦ //        block   2 : ordering dp
↦ //                  : names    s(1)(1) s(1)(2) s(2)(1) s(2)(2) x y Dx Dy
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    s(1)(2)s(1)(1)=s(1)(1)*s(1)(2)-s(1)(2)
↦ //    s(2)(1)s(1)(1)=s(1)(1)*s(2)(1)+s(2)(1)
↦ //    s(2)(1)s(1)(2)=s(1)(2)*s(2)(1)-s(1)(1)+s(2)(2)
↦ //    s(2)(2)s(1)(2)=s(1)(2)*s(2)(2)-s(1)(2)
↦ //    s(2)(2)s(2)(1)=s(2)(1)*s(2)(2)+s(2)(1)
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
LD;
↦ LD[1]=5*s(2)(2)-y*Dy
↦ LD[2]=3*s(1)(1)-x*Dx
↦ LD[3]=15*s(1)(2)*s(2)(1)-x*y*Dx*Dy-5*x*Dx
↦ LD[4]=5*s(2)(1)*y^4-x^3*Dy
↦ LD[5]=3*s(1)(2)*x^2-y^5*Dx
```

7.5.7.4 makeMalgrange

Procedure from library `dmodvar.lib` (see Section 7.5.7 [dmodvar_lib], page 453).

Usage: makeMalgrange(F [,ORD]); F an ideal, ORD an optional string

Return: ring (Weyl algebra) containing an ideal IF

Purpose: create the ideal by Malgrange associated with F = F[1],...,F[P].

Note: Activate the output ring with the `setring` command. In this ring, the ideal IF is the ideal by Malgrange corresponding to F.
The value of ORD must be an arbitrary ordering in K<_t,_x,_Dt,_Dx> written in the string form. By default ORD = 'dp'.

Display: If printlevel=1, progress debug messages will be printed,
if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodvar.lib";
ring R = 0,(x,y,z),Dp;
ideal I = x^2+y^3, z;
def W = makeMalgrange(I);
setring W;
W;
↦ // coefficients: QQ
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    t(1) t(2) x y z Dt(1) Dt(2) Dx Dy Dz
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dt(1)t(1)=t(1)*Dt(1)+1
↦ //    Dt(2)t(2)=t(2)*Dt(2)+1
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
↦ //    Dzz=z*Dz+1
IF;
↦ IF[1]=-y^3-x^2+t(1)
↦ IF[2]=t(2)-z
↦ IF[3]=2*x*Dt(1)+Dx
↦ IF[4]=3*y^2*Dt(1)+Dy
↦ IF[5]=Dt(2)+Dz
```

7.5.8 involut_lib

Library: involut.lib

Purpose: Computations and operations with involutions

Authors: Oleksandr Iena, yena@mathematik.uni-kl.de,
Markus Becker, mbecker@mathematik.uni-kl.de,
Viktor Levandovskyy, levandov@mathematik.uni-kl.de

Overview: Involution is an anti-automorphism of a non-commutative K-algebra with the property that applied an involution twice, one gets an identity. Involution is linear with respect to the ground field. In this library we compute linear involutions, distinguishing the case of a diagonal matrix (such involutions are called homothetic) and a general one. Also, linear automorphisms of different order can be computed.

Support: Forschungsschwerpunkt 'Mathematik und Praxis' (Project of Dr. E. Zerz and V. Levandovskyy), Uni Kaiserslautern

Remark: This library provides algebraic tools for computations and operations with algebraic involutions and linear automorphisms of non-commutative algebras

Procedures:

7.5.8.1 findInvo

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: findInvo();

Return: a ring containing a list L of pairs, where
L[i][1] = ideal; a Groebner Basis of an i-th associated prime,
L[i][2] = matrix, defining a linear map, with entries, reduced with respect to L[i][1]

Purpose: computed the ideal of linear involutions of the basering

Assume: the relations on the algebra are of the form YX = XY + D, that is the current ring is a G-algebra of Lie type.

Note: for convenience, the full ideal of relations `idJ` and the initial matrix with indeterminates `matD` are exported in the output ring

Example:

```
LIB "involut.lib";
def a = makeWeyl(1);
setring a; // this algebra is a first Weyl algebra
a;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x D
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dx=xD+1
def X = findInvo();
setring X; // ring with new variables, corr. to unknown coefficients
X;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    a11 a12 a21 a22
↦ //        block   2 : ordering C
```

```
L;
↦ [1]:
↦    [1]:
↦       _[1]=a11+a22
↦       _[2]=a12*a21+a22^2-1
↦    [2]:
↦       _[1,1]=-a22
↦       _[1,2]=a12
↦       _[2,1]=a21
↦       _[2,2]=a22
// look at the matrix in the new variables, defining the linear involution
print(L[1][2]);
↦ -a22,a12,
↦ a21, a22
L[1][1];  // where new variables obey these relations
↦ _[1]=a11+a22
↦ _[2]=a12*a21+a22^2-1
idJ;
↦ idJ[1]=-a12*a21+a11*a22+1
↦ idJ[2]=a11^2+a12*a21-1
↦ idJ[3]=a11*a12+a12*a22
↦ idJ[4]=a11*a21+a21*a22
↦ idJ[5]=a12*a21+a22^2-1
```

See also: Section 7.5.8.2 [findInvoDiag], page 458; Section 7.5.8.5 [involution], page 461.

7.5.8.2 findInvoDiag

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: findInvoDiag();

Return: a ring together with a list of pairs L, where
L[i][1] = ideal; a Groebner Basis of an i-th associated prime,
L[i][2] = matrix, defining a linear map, with entries, reduced with respect to L[i][1]

Purpose: compute homothetic (diagonal) involutions of the basering

Assume: the relations on the algebra are of the form YX = XY + D, that is the current ring is a G-algebra of Lie type.

Note: for convenience, the full ideal of relations `idJ` and the initial matrix with indeterminates `matD` are exported in the output ring

Example:

```
LIB "involut.lib";
def a = makeWeyl(1);
setring a; // this algebra is a first Weyl algebra
a;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x D
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dx=xD+1
```

```
def X = findInvoDiag();
setring X; // ring with new variables, corresponding to unknown coefficients
X;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    a11 a22
↦ //        block   2 : ordering C
// print matrices, defining linear involutions
print(L[1][2]);  // a first matrix: we see it is constant
↦ -1,0,
↦ 0, 1
print(L[2][2]);  // and a second possible matrix; it is constant too
↦ 1,0,
↦ 0,-1
L; // let us take a look on the whole list
↦ [1]:
↦    [1]:
↦       _[1]=a22-1
↦       _[2]=a11+1
↦    [2]:
↦       _[1,1]=-1
↦       _[1,2]=0
↦       _[2,1]=0
↦       _[2,2]=1
↦ [2]:
↦    [1]:
↦       _[1]=a22+1
↦       _[2]=a11-1
↦    [2]:
↦       _[1,1]=1
↦       _[1,2]=0
↦       _[2,1]=0
↦       _[2,2]=-1
idJ;
↦ idJ[1]=a11*a22+1
↦ idJ[2]=a11^2-1
↦ idJ[3]=a22^2-1
```

See also: Section 7.5.8.1 [findInvo], page 457; Section 7.5.8.5 [involution], page 461.

7.5.8.3 findAuto

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: findAuto(n); n an integer

Return: a ring together with a list of pairs L, where
L[i][1] = ideal; a Groebner Basis of an i-th associated prime,
L[i][2] = matrix, defining a linear map, with entries, reduced with respect to L[i][1]

Purpose: compute the ideal of linear automorphisms of the basering,
given by a matrix, n-th power of which gives identity (i.e. unipotent matrix)

Assume: the relations on the algebra are of the form YX = XY + D, that is the current ring is a G-algebra of Lie type.

Note: if n=0, a matrix, defining an automorphism is not assumed to be unipotent but just non-degenerate. A nonzero parameter `@p` is introduced as the value of the determinant of the matrix above.
For convenience, the full ideal of relations `idJ` and the initial matrix with indeterminates
`matD` are mutually exported in the output ring

Example:

```
LIB "involut.lib";
def a = makeWeyl(1);
setring a; // this algebra is a first Weyl algebra
a;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x D
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dx=xD+1
def X = findAuto(2);  // in contrast to findInvo look for automorphisms
setring X; // ring with new variables - unknown coefficients
X;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    a11 a12 a21 a22
↦ //        block   2 : ordering C
size(L); // we have (size(L)) families in the answer
↦ 2
// look at matrices, defining linear automorphisms:
print(L[1][2]);  // a first one: we see it is the identity
↦ 1,0,
↦ 0,1
print(L[2][2]);  // and a second possible matrix; it is diagonal
↦ -1,0,
↦ 0, -1
// L; // we can take a look on the whole list, too
idJ;
↦ idJ[1]=-a12*a21+a11*a22-1
↦ idJ[2]=a11^2+a12*a21-1
↦ idJ[3]=a11*a12+a12*a22
↦ idJ[4]=a11*a21+a21*a22
↦ idJ[5]=a12*a21+a22^2-1
kill X; kill a;
//----------- find all the linear automorphisms --------------------
//----------- use the call findAuto(0)          --------------------
ring R = 0,(x,s),dp;
def r = nc_algebra(1,s); setring r; // the shift algebra
s*x; // the only relation in the algebra is:
↦ xs+s
def Y = findAuto(0);
setring Y;
size(L); // here, we have 1 parametrized family
```

```
↦ 1
print(L[1][2]); // here, @p is a nonzero parameter
↦ 1,a12,
↦ 0,(@p)
det(L[1][2]-@p);  // check whether determinante is zero
↦ 0
```

See also: Section 7.5.8.1 [findInvo], page 457.

7.5.8.4 ncdetection

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: ncdetection();

Return: ideal, representing an involution map

Purpose: compute classical involutions (i.e. acting rather on operators than on variables) for some particular noncommutative algebras

Assume: the procedure is aimed at non-commutative algebras with differential, shift or advance operators arising in Control Theory. It has to be executed in a ring.

Example:

```
LIB "involut.lib";
ring R = 0,(x,y,z,D(1..3)),dp;
matrix D[6][6];
D[1,4]=1; D[2,5]=1;  D[3,6]=1;
def r = nc_algebra(1,D); setring r;
ncdetection();
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦ _[4]=-D(1)
↦ _[5]=-D(2)
↦ _[6]=-D(3)
kill r, R;
//----------------------------------------
ring R=0,(x,S),dp;
def r = nc_algebra(1,-S); setring r;
ncdetection();
↦ _[1]=-x
↦ _[2]=S
kill r, R;
//----------------------------------------
ring R=0,(x,D(1),S),dp;
matrix D[3][3];
D[1,2]=1;  D[1,3]=-S;
def r = nc_algebra(1,D); setring r;
ncdetection();
↦ _[1]=-x
↦ _[2]=D(1)
↦ _[3]=S
```

7.5.8.5 involution

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: involution(m, theta); m is a poly/vector/ideal/matrix/module, theta is a map

Return: object of the same type as m

Purpose: applies the involution, presented by theta to the object m

Theory: for an involution theta and two polynomials a,b from the algebra,
theta(ab) = theta(b) theta(a); theta is linear with respect to the ground field

Note: This is generalized "theta(m)" for data types unsupported by "map".

Example:

```
LIB "involut.lib";
ring R = 0,(x,d),dp;
def r = nc_algebra(1,1); setring r; // Weyl-Algebra
map F = r,x,-d;
F(F);  // should be maxideal(1) for an involution
↦ _[1]=x
↦ _[2]=d
poly f =  x*d^2+d;
poly If = involution(f,F);
f-If;
↦ 0
poly g = x^2*d+2*x*d+3*x+7*d;
poly tg = -d*x^2-2*d*x+3*x-7*d;
poly Ig = involution(g,F);
tg-Ig;
↦ 0
ideal I = f,g;
ideal II = involution(I,F);
II;
↦ II[1]=xd2+d
↦ II[2]=-x2d-2xd+x-7d-2
matrix(I) - involution(II,F);
↦ _[1,1]=0
↦ _[1,2]=0
module M  = [f,g,0],[g,0,x^2*d];
module IM = involution(M,F);
print(IM);
↦ xd2+d,          -x2d-2xd+x-7d-2,
↦ -x2d-2xd+x-7d-2,0,
↦ 0,              -x2d-2x
print(matrix(M) - involution(IM,F));
↦ 0,0,
↦ 0,0,
↦ 0,0
```

7.5.8.6 isInvolution

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: isInvolution(F); F is a map from current ring to itself

Return: integer, 1 if F determines an involution and 0 otherwise

Theory: involution is an antiautomorphism of order 2

Assume: F is a map from current ring to itself

Example:

```
LIB "involut.lib";
def A = makeUsl(2); setring A;
map I = A,-e,-f,-h; //correct antiauto involution
isInvolution(I);
↦ 1
map J = A,3*e,1/3*f,-h; // antiauto but not involution
isInvolution(J);
↦ 0
map K = A,f,e,-h; // not antiauto
isInvolution(K);
↦ 0
```

See also: Section 7.5.8.1 [findInvo], page 457; Section 7.5.8.5 [involution], page 461; Section 7.5.8.7 [isAntiEndo], page 463.

7.5.8.7 isAntiEndo

Procedure from library `involut.lib` (see Section 7.5.8 [involut_lib], page 456).

Usage: isAntiEndo(F); F is a map from current ring to itself

Return: integer, 1 if F determines an antiendomorphism of
current ring and 0 otherwise

Assume: F is a map from current ring to itself

Example:

```
LIB "involut.lib";
def A = makeUsl(2); setring A;
map I = A,-e,-f,-h; //correct antiauto involution
isAntiEndo(I);
↦ 1
map J = A,3*e,1/3*f,-h; // antiauto but not involution
isAntiEndo(J);
↦ 1
map K = A,f,e,-h; // not antiendo
isAntiEndo(K);
↦ 0
```

See also: Section 7.5.8.1 [findInvo], page 457; Section 7.5.8.5 [involution], page 461; Section 7.5.8.6 [isInvolution], page 462.

7.5.9 gkdim_lib

Library: gkdim.lib

Purpose: Procedures for calculating the Gelfand-Kirillov dimension

Authors: Lobillo, F.J., jlobillo@ugr.es
Rabelo, C., crabelo@ugr.es

Support: 'Metodos algebraicos y efectivos en grupos cuanticos', BFM2001-3141, MCYT, Jose Gomez-Torrecillas (Main researcher).

Note: The built-in command `dim`, executed for a module in @plural, computes the Gelfand-Kirillov dimension.

Procedures:

7.5.9.1 GKdim

Procedure from library `gkdim.lib` (see Section 7.5.9 [gkdim_lib], page 463).

Usage: GKdim(L); L is a left ideal/module/matrix

Return: int

Purpose: compute the Gelfand-Kirillov dimension of the factor-module, whose presentation is given by L, e.g. R^r/L

Note: if the factor-module is zero, -1 is returned

Example:

```
LIB "gkdim.lib";
ring R = 0,(x,y,z),Dp;
matrix C[3][3]=0,1,1,0,0,-1,0,0,0;
matrix D[3][3]=0,0,0,0,0,x;
def r = nc_algebra(C,D); setring r;
r;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering Dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    zy=-yz+x
ideal I=x;
GKdim(I);
↦ 2
ideal J=x2,y;
GKdim(J);
↦ 1
module M=[x2,y,1],[x,y2,0];
GKdim(M);
↦ 3
ideal A = x,y,z;
GKdim(A);
↦ 0
ideal B = 1;
GKdim(B);
↦ -1
GKdim(ideal(0)) == nvars(basering);  // should be true, i.e., evaluated to 1
↦ 1
```

7.5.10 ncalg_lib

Library: ncalg.lib

Purpose: Definitions of important G- and GR-algebras

Authors: Viktor Levandovskyy, levandov@mathematik.uni-kl.de,
Oleksandr Motsak, U@D, where U={motsak}, D={mathematik.uni-kl.de}

Conventions:
This library provides pre-defined important noncommutative algebras.
For universal enveloping algebras of finite dimensional Lie algebras sl_n, gl_n, g_2 etc.

there are functions `makeUsl`, `makeUgl`, `makeUg2` etc.
For quantized enveloping algebras U_q(sl_2) and U_q(sl_3), there are functions `makeQsl2`, `makeQsl3`) and for non-standard quantum deformation of so_3, there is the function `makeQso3`.
For bigger algebras we suppress the output of the (lengthy) list of non-commutative relations and provide only the number of these relations instead.

Procedures:

7.5.10.1 makeUsl2

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsl2([p]), p an optional integer (field characteristic)

Return: ring

Purpose: set up the U(sl_2) in the variables e,f,h over the field of char p

Note: activate this ring with the `setring` command

Example:

```
LIB "ncalg.lib";
def a=makeUsl2();
setring a;
a;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
```

See also: Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.3 [makeUgl], page 466; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.2 makeUsl

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsl(n,[p]); n an integer, n>1; p an optional integer (field characteristic)

Return: ring

Purpose: set up the U(sl_n) in the variables (x(i),y(i),h(i) | i=1..n+1) over the field of char p

Note: activate this ring with the `setring` command
This presentation of U(sl_n) is the standard one, i.e. positive resp. negative roots are denoted by x(i) resp. y(i) and the Cartan elements are denoted by h(i).
The variables are ordered as x(1),...x(n),y(1),...,y(n),h(1),...h(n).

Example:

```
LIB "ncalg.lib";
def a=makeUsl(3);
setring a;
```

```
a;
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) y(1) y(2) y(3) h(1) h(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(2)x(1)=x(1)*x(2)+x(3)
↦ //    y(1)x(1)=x(1)*y(1)-h(1)
↦ //    y(3)x(1)=x(1)*y(3)-y(2)
↦ //    h(1)x(1)=x(1)*h(1)+2*x(1)
↦ //    h(2)x(1)=x(1)*h(2)-x(1)
↦ //    y(2)x(2)=x(2)*y(2)-h(2)
↦ //    y(3)x(2)=x(2)*y(3)+y(1)
↦ //    h(1)x(2)=x(2)*h(1)-x(2)
↦ //    h(2)x(2)=x(2)*h(2)+2*x(2)
↦ //    y(1)x(3)=x(3)*y(1)-x(2)
↦ //    y(2)x(3)=x(3)*y(2)+x(1)
↦ //    y(3)x(3)=x(3)*y(3)-h(1)-h(2)
↦ //    h(1)x(3)=x(3)*h(1)+x(3)
↦ //    h(2)x(3)=x(3)*h(2)+x(3)
↦ //    y(2)y(1)=y(1)*y(2)-y(3)
↦ //    h(1)y(1)=y(1)*h(1)-2*y(1)
↦ //    h(2)y(1)=y(1)*h(2)+y(1)
↦ //    h(1)y(2)=y(2)*h(1)+y(2)
↦ //    h(2)y(2)=y(2)*h(2)-2*y(2)
↦ //    h(1)y(3)=y(3)*h(1)-y(3)
↦ //    h(2)y(3)=y(3)*h(2)-y(3)
```

See also: Section 7.5.10.24 [makeQsl3], page 479; Section 7.5.10.22 [makeQso3], page 478; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.3 [makeUgl], page 466; Section 7.5.10.1 [makeUsl2], page 465.

7.5.10.3 makeUgl

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUgl(n,[p]); n an int, n>1; p an optional int (field characteristic)

Return: ring

Purpose: set up the U(gl_n) in the (e_ij (1<i,j<n)) presentation (where e_ij corresponds to a matrix with 1 at i,j only) over the field of char p

Note: activate this ring with the `setring` command
the variables are ordered as e_12,e_13,...,e_1n,e_21,...,e_nn.

Example:

```
LIB "ncalg.lib";
def a=makeUgl(3);
setring a; a;
↦ // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    e_1_1 e_1_2 e_1_3 e_2_1 e_2_2 e_2_3 e_3_1 \
   e_3_2 e_3_3
```

```
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    e_1_2e_1_1=e_1_1*e_1_2-e_1_2
↦ //    e_1_3e_1_1=e_1_1*e_1_3-e_1_3
↦ //    e_2_1e_1_1=e_1_1*e_2_1+e_2_1
↦ //    e_3_1e_1_1=e_1_1*e_3_1+e_3_1
↦ //    e_2_1e_1_2=e_1_2*e_2_1-e_1_1+e_2_2
↦ //    e_2_2e_1_2=e_1_2*e_2_2-e_1_2
↦ //    e_2_3e_1_2=e_1_2*e_2_3-e_1_3
↦ //    e_3_1e_1_2=e_1_2*e_3_1+e_3_2
↦ //    e_2_1e_1_3=e_1_3*e_2_1+e_2_3
↦ //    e_3_1e_1_3=e_1_3*e_3_1-e_1_1+e_3_3
↦ //    e_3_2e_1_3=e_1_3*e_3_2-e_1_2
↦ //    e_3_3e_1_3=e_1_3*e_3_3-e_1_3
↦ //    e_2_2e_2_1=e_2_1*e_2_2+e_2_1
↦ //    e_3_2e_2_1=e_2_1*e_3_2+e_3_1
↦ //    e_2_3e_2_2=e_2_2*e_2_3-e_2_3
↦ //    e_3_2e_2_2=e_2_2*e_3_2+e_3_2
↦ //    e_3_1e_2_3=e_2_3*e_3_1-e_2_1
↦ //    e_3_2e_2_3=e_2_3*e_3_2-e_2_2+e_3_3
↦ //    e_3_3e_2_3=e_2_3*e_3_3-e_2_3
↦ //    e_3_3e_3_1=e_3_1*e_3_3+e_3_1
↦ //    e_3_3e_3_2=e_3_2*e_3_3+e_3_2
```

See also: Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.4 makeUso5

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso5([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_5)

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_5) is derived from the Chevalley representation of so_5, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso5();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) Y(1) Y(2) Y(3) Y(4) H(\
   1) H(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  28  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.5 makeUso6

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso6([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_6)

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_6) is derived from the Chevalley representation of so_6, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso6();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 15
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) Y(1) Y(2) Y(\
   3) Y(4) Y(5) Y(6) H(1) H(2) H(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  60  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.6 makeUso7

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso7([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_7)

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_7) is derived from the Chevalley representation of so_7, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso7();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 21
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) H(1) H(2) H(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  107  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.7 makeUso8

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso8([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_8)

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_8) is derived from the Chevalley representation of so_8, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso8();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 28
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y\
   (11) Y(12) H(1) H(2) H(3) H(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  180  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.8 makeUso9

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso9([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_9)

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_9) is derived from the Chevalley representation of so_9, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso9();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 36
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6\
   ) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) H(1) H(2) H(3)\
    H(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  264  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.9 makeUso10

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso10([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_{10})

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_{10}) is derived from the Chevalley representation of so_{10}, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso10();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 45
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) Y(1)\
    Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(\
   15) Y(16) Y(17) Y(18) Y(19) Y(20) H(1) H(2) H(3) H(4) H(5)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  390  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.10 makeUso11

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso11([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_{11})

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_{11}) is derived from the Chevalley representation of so_{11}, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso11();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 55
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(\
   10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21) Y(2\
   2) Y(23) Y(24) Y(25) H(1) H(2) H(3) H(4) H(5)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  523  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.11 makeUso12

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUso12([p]); p an optional integer (field characteristic)

Return: a ring, describing U(so_{12})

Note: You have to activate this ring with the 'setring' command. The presentation of U(so_{12}) is derived from the Chevalley representation of so_{12}, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUso12();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 66
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) X(26) X(27) X(28) X(29) X(30) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29\
   ) Y(30) H(1) H(2) H(3) H(4) H(5) H(6)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  714  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.12 makeUsp1

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsp1([p]); p an optional integer (field characteristic)

Return: a ring, describing U(sp_1)

Note: You have to activate this ring with the 'setring' command. The presentation of U(sp_1) is derived from the Chevalley representation of sp_1, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUsp1();
setring ncAlgebra;
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) Y(1) H(1)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Y(1)X(1)=X(1)*Y(1)-H(1)
↦ //    H(1)X(1)=X(1)*H(1)+2*X(1)
↦ //    H(1)Y(1)=Y(1)*H(1)-2*Y(1)
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467.

7.5.10.13 makeUsp2

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsp2([p]); p an optional integer (field characteristic)

Return: a ring, describing U(sp_2)

Note: You have to activate this ring with the 'setring' command. The presentation of U(sp_2) is derived from the Chevalley representation of sp_2, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUsp2();
setring ncAlgebra;
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) Y(1) Y(2) Y(3) Y(4) H(\
   1) H(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    X(2)X(1)=X(1)*X(2)+X(3)
↦ //    X(3)X(1)=X(1)*X(3)+2*X(4)
↦ //    Y(1)X(1)=X(1)*Y(1)-H(1)
↦ //    Y(3)X(1)=X(1)*Y(3)-2*Y(2)
↦ //    Y(4)X(1)=X(1)*Y(4)-Y(3)
↦ //    H(1)X(1)=X(1)*H(1)+2*X(1)
↦ //    H(2)X(1)=X(1)*H(2)-X(1)
↦ //    Y(2)X(2)=X(2)*Y(2)-H(2)
↦ //    Y(3)X(2)=X(2)*Y(3)+Y(1)
↦ //    H(1)X(2)=X(2)*H(1)-2*X(2)
↦ //    H(2)X(2)=X(2)*H(2)+2*X(2)
↦ //    Y(1)X(3)=X(3)*Y(1)-2*X(2)
↦ //    Y(2)X(3)=X(3)*Y(2)+X(1)
↦ //    Y(3)X(3)=X(3)*Y(3)-H(1)-2*H(2)
↦ //    Y(4)X(3)=X(3)*Y(4)+Y(1)
↦ //    H(2)X(3)=X(3)*H(2)+X(3)
↦ //    Y(1)X(4)=X(4)*Y(1)-X(3)
↦ //    Y(3)X(4)=X(4)*Y(3)+X(1)
↦ //    Y(4)X(4)=X(4)*Y(4)-H(1)-H(2)
↦ //    H(1)X(4)=X(4)*H(1)+2*X(4)
↦ //    Y(2)Y(1)=Y(1)*Y(2)-Y(3)
↦ //    Y(3)Y(1)=Y(1)*Y(3)-2*Y(4)
↦ //    H(1)Y(1)=Y(1)*H(1)-2*Y(1)
↦ //    H(2)Y(1)=Y(1)*H(2)+Y(1)
↦ //    H(1)Y(2)=Y(2)*H(1)+2*Y(2)
↦ //    H(2)Y(2)=Y(2)*H(2)-2*Y(2)
↦ //    H(2)Y(3)=Y(3)*H(2)-Y(3)
↦ //    H(1)Y(4)=Y(4)*H(1)-2*Y(4)
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.14 makeUsp3

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsp3([p]); p an optional integer (field characteristic)

Return: a ring, describing U(sp_3)

Note: You have to activate this ring with the 'setring' command. The presentation of U(sp_3) is derived from the Chevalley representation of sp_3, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUsp3();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 21
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) H(1) H(2) H(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
```

```
setring ncAlgebra;
// ...  107  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.15 makeUsp4

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsp4([p]); p an optional integer (field characteristic)

Return: a ring, describing U(sp_4)

Note: You have to activate this ring with the 'setring' command. The presentation of U(sp_4) is derived from the Chevalley representation of sp_4, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUsp4();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 36
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6\
   ) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) H(1) H(2) H(3)\
    H(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  264  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.16 makeUsp5

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUsp5([p]); p an optional integer (field characteristic)

Return: a ring, describing U(sp_5)

Note: You have to activate this ring with the 'setring' command. The presentation of U(sp_5) is derived from the Chevalley representation of sp_5, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUsp5();
ncAlgebra;
```

```
↦ // coefficients: QQ
↦ // number of vars : 55
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(\
   10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21) Y(2\
   2) Y(23) Y(24) Y(25) H(1) H(2) H(3) H(4) H(5)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  523  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.17 makeUg2

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUg2([p]), p an optional int (field characteristic)

Return: ring

Purpose: set up the U(g_2) in variables (x(i),y(i),Ha,Hb) for i=1..6 over the field of char p

Note: activate this ring with the `setring` command
the variables are ordered as x(1),...x(6),y(1),...,y(6),Ha,Hb.

Example:

```
LIB "ncalg.lib";
def a = makeUg2();
a;
↦ // coefficients: QQ
↦ // number of vars : 14
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4) x(5) x(6) y(1) y(2) y(\
   3) y(4) y(5) y(6) Ha Hb
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring a;
// ...  56  noncommutative relations
```

See also: Section 7.5.10.3 [makeUgl], page 466; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.18 makeUf4

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUf4([p]); p an optional integer (field characteristic)

Return: a ring, describing U(f_4)

Note: You have to activate this ring with the 'setring' command. The presentation of U(f_4) is derived from the Chevalley representation of f_4, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUf4();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 52
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(\
   11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21) Y(22) Y(2\
   3) Y(24) H(1) H(2) H(3) H(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  552  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.19 makeUe6

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUe6([p]); p an optional integer (field characteristic)

Return: a ring, describing U(e_6)

Note: You have to activate this ring with the 'setring' command. The presentation of U(e_6) is derived from the Chevalley representation of e_6, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUe6();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 78
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) X(26) X(27) X(28) X(29) X(30) X(31) X(32) X(33)\
    X(34) X(35) X(36) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(1\
   1) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23\
   ) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29) Y(30) Y(31) Y(32) Y(33) Y(34) Y(35)\
    Y(36) H(1) H(2) H(3) H(4) H(5) H(6)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  1008  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.20 makeUe7

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUe7([p]); p an optional integer (field characteristic)

Return: a ring, describing U(e_7)

Note: You have to activate this ring with the 'setring' command. The presentation of U(e_7) is derived from the Chevalley representation of e_7, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUe7();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 133
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) X(26) X(27) X(28) X(29) X(30) X(31) X(32) X(33)\
    X(34) X(35) X(36) X(37) X(38) X(39) X(40) X(41) X(42) X(43) X(44) X(45) \
   X(46) X(47) X(48) X(49) X(50) X(51) X(52) X(53) X(54) X(55) X(56) X(57) X\
   (58) X(59) X(60) X(61) X(62) X(63) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8\
   ) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) \
   Y(21) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29) Y(30) Y(31) Y(32) Y\
   (33) Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41) Y(42) Y(43) Y(44) Y(\
   45) Y(46) Y(47) Y(48) Y(49) Y(50) Y(51) Y(52) Y(53) Y(54) Y(55) Y(56) Y(5\
   7) Y(58) Y(59) Y(60) Y(61) Y(62) Y(63) H(1) H(2) H(3) H(4) H(5) H(6) H(7)
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  2541  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.21 makeUe8

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeUe8([p]); p an optional integer (field characteristic)

Return: a ring, describing U(e_8)

Note: You have to activate this ring with the 'setring' command. The presentation of U(e_8) is derived from the Chevalley representation of e_8, positive resp. negative roots are denoted by x(i) resp. y(i); Cartan elements are denoted by h(i).

Example:

```
LIB "ncalg.lib";
def ncAlgebra = makeUe8();
ncAlgebra;
↦ // coefficients: QQ
↦ // number of vars : 248
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) X(8) X(\
   9) X(10) X(11) X(12) X(13) X(14) X(15) X(16) X(17) X(18) X(19) X(20) X(21\
   ) X(22) X(23) X(24) X(25) X(26) X(27) X(28) X(29) X(30) X(31) X(32) X(33)\
    X(34) X(35) X(36) X(37) X(38) X(39) X(40) X(41) X(42) X(43) X(44) X(45) \
   X(46) X(47) X(48) X(49) X(50) X(51) X(52) X(53) X(54) X(55) X(56) X(57) X\
   (58) X(59) X(60) X(61) X(62) X(63) X(64) X(65) X(66) X(67) X(68) X(69) X(\
   70) X(71) X(72) X(73) X(74) X(75) X(76) X(77) X(78) X(79) X(80) X(81) X(8\
   2) X(83) X(84) X(85) X(86) X(87) X(88) X(89) X(90) X(91) X(92) X(93) X(94\
   ) X(95) X(96) X(97) X(98) X(99) X(100) X(101) X(102) X(103) X(104) X(105)\
    X(106) X(107) X(108) X(109) X(110) X(111) X(112) X(113) X(114) X(115) X(\
   116) X(117) X(118) X(119) X(120) Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) \
   Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(\
   21) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29) Y(30) Y(31) Y(32) Y(3\
   3) Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41) Y(42) Y(43) Y(44) Y(45\
   ) Y(46) Y(47) Y(48) Y(49) Y(50) Y(51) Y(52) Y(53) Y(54) Y(55) Y(56) Y(57)\
    Y(58) Y(59) Y(60) Y(61) Y(62) Y(63) Y(64) Y(65) Y(66) Y(67) Y(68) Y(69) \
   Y(70) Y(71) Y(72) Y(73) Y(74) Y(75) Y(76) Y(77) Y(78) Y(79) Y(80) Y(81) Y\
   (82) Y(83) Y(84) Y(85) Y(86) Y(87) Y(88) Y(89) Y(90) Y(91) Y(92) Y(93) Y(\
   94) Y(95) Y(96) Y(97) Y(98) Y(99) Y(100) Y(101) Y(102) Y(103) Y(104) Y(10\
   5) Y(106) Y(107) Y(108) Y(109) Y(110) Y(111) Y(112) Y(113) Y(114) Y(115) \
   Y(116) Y(117) Y(118) Y(119) Y(120) H(1) H(2) H(3) H(4) H(5) H(6) H(7) H(8\
   )
↦ //        block   2 : ordering C
↦ // noncommutative relations: ...
setring ncAlgebra;
// ...  7752  noncommutative relations
```

See also: Section 7.5.10.19 [makeUe6], page 476; Section 7.5.10.20 [makeUe7], page 477; Section 7.5.10.21 [makeUe8], page 477; Section 7.5.10.18 [makeUf4], page 475; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.2 [makeUsl], page 465; Section 7.5.10.4 [makeUso5], page 467; Section 7.5.10.12 [makeUsp1], page 472.

7.5.10.22 makeQso3

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeQso3([n]), n an optional int

Purpose: set up the U_q(so_3) in the presentation of Klimyk; if n is specified, the quantum parameter Q will be specialized at the (2n)-th root of unity

Return: ring

Note: activate this ring with the `setring` command

Example:

```
LIB "ncalg.lib";
def K = makeQso3(3);
setring K;
K;
↦ // coefficients: QQ[Q]/(Q2-Q+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(Q-1)*xy+(-Q)*z
↦ //    zx=(-Q)*xz+(-Q+1)*y
↦ //    zy=(Q-1)*yz+(-Q)*x
```

See also: Section 7.5.10.25 [Qso3Casimir], page 481; Section 7.5.10.23 [makeQsl2], page 479; Section 7.5.10.24 [makeQsl3], page 479; Section 7.5.10.17 [makeUg2], page 475; Section 7.5.10.3 [makeUgl], page 466; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.23 makeQsl2

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeQsl2([n]), n an optional int

Return: ring

Purpose: define the U_q(sl_2) as a factor-ring of a ring V_q(sl_2) modulo the ideal `Qideal`

Note: the output consists of a ring, presenting V_q(sl_2) together with the ideal called `Qideal` in this ring
activate this ring with the `setring` command
in order to create the U_q(sl_2) from the output, execute the command like `qring Usl2q = Qideal;`
If n is specified, the quantum parameter q will be specialized at the n-th root of unity

Example:

```
LIB "ncalg.lib";
def A = makeQsl2(3);
setring A;
Qideal;
↦ Qideal[1]=Ke*Kf-1
qring Usl2q = Qideal;
Usl2q;
↦ // coefficients: QQ[q]/(q^2+q+1)
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    E F Ke Kf
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    FE=E*F+(2/3*q+1/3)*Ke+(-2/3*q-1/3)*Kf
↦ //    KeE=(-q-1)*E*Ke
↦ //    KfE=(q)*E*Kf
↦ //    KeF=(q)*F*Ke
↦ //    KfF=(-q-1)*F*Kf
↦ // quotient ring from ideal
↦ _[1]=Ke*Kf-1
```

See also: Section 7.5.10.24 [makeQsl3], page 479; Section 7.5.10.22 [makeQso3], page 478; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.24 makeQsl3

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: makeQsl3([n]), n an optional int

Return: ring

Purpose: define the U_q(sl_3) as a factor-ring of a ring V_q(sl_3) modulo the ideal `Qideal`

Note: the output consists of a ring, presenting V_q(sl_3) together with the ideal called `Qideal` in this ring
activate this ring with the `setring` command
in order to create the U_q(sl_3) from the output, execute the command like `qring Usl3q = Qideal;`
If n is specified, the quantum parameter q will be specialized at the n-th root of unity

Example:

```
LIB "ncalg.lib";
def B = makeQsl3(5);
setring B;
qring Usl3q = Qideal;
Usl3q;
↦ // coefficients: QQ[q]/(q^4+q^3+q^2+q+1)
↦ // number of vars : 10
↦ //        block   1 : ordering wp
↦ //                  : names    f12 f13 f23 k1 k2 l1 l2 e12 e13 e23
↦ //                  : weights    2   3   2   1   1   1   1   2   3   2
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    f13f12=(q^3)*f12*f13
↦ //    f23f12=(q^2)*f12*f23+(-q)*f13
↦ //    k1f12=(q^3)*f12*k1
↦ //    k2f12=(q)*f12*k2
↦ //    l1f12=(q^2)*f12*l1
↦ //    l2f12=(-q^3-q^2-q-1)*f12*l2
↦ //    e12f12=f12*e12+(1/5*q^3-3/5*q^2-2/5*q-1/5)*k1^2+(-1/5*q^3+3/5*q^2+2\
   /5*q+1/5)*l1^2
↦ //    e13f12=f12*e13+(q^3+q^2+q+1)*l1^2*e23
↦ //    f23f13=(q^3)*f13*f23
↦ //    k1f13=(-q^3-q^2-q-1)*f13*k1
↦ //    k2f13=(-q^3-q^2-q-1)*f13*k2
↦ //    l1f13=(q)*f13*l1
↦ //    l2f13=(q)*f13*l2
↦ //    e12f13=f13*e12+(q)*f23*k1^2
↦ //    e13f13=f13*e13+(-1/5*q^3+3/5*q^2+2/5*q+1/5)*k1^2*k2^2+(1/5*q^3-3/5*\
   q^2-2/5*q-1/5)*l1^2*l2^2
↦ //    e23f13=f13*e23+(q^3+q^2+q+1)*f12*l2^2
↦ //    k1f23=(q)*f23*k1
↦ //    k2f23=(q^3)*f23*k2
↦ //    l1f23=(-q^3-q^2-q-1)*f23*l1
↦ //    l2f23=(q^2)*f23*l2
↦ //    e13f23=f23*e13+(q)*k2^2*e12
↦ //    e23f23=f23*e23+(1/5*q^3-3/5*q^2-2/5*q-1/5)*k2^2+(-1/5*q^3+3/5*q^2+2\
   /5*q+1/5)*l2^2
↦ //    e12k1=(q^3)*k1*e12
↦ //    e13k1=(-q^3-q^2-q-1)*k1*e13
↦ //    e23k1=(q)*k1*e23
↦ //    e12k2=(q)*k2*e12
↦ //    e13k2=(-q^3-q^2-q-1)*k2*e13
↦ //    e23k2=(q^3)*k2*e23
↦ //    e12l1=(q^2)*l1*e12
↦ //    e13l1=(q)*l1*e13
↦ //    e23l1=(-q^3-q^2-q-1)*l1*e23
↦ //    e12l2=(-q^3-q^2-q-1)*l2*e12
↦ //    e13l2=(q)*l2*e13
↦ //    e23l2=(q^2)*l2*e23
↦ //    e13e12=(q^3)*e12*e13
↦ //    e23e12=(q^2)*e12*e23+(-q)*e13
↦ //    e23e13=(q^3)*e13*e23
↦ // quotient ring from ideal
↦ _[1]=k2*l2-1
↦ _[2]=k1*l1-1
```

See also: Section 7.5.10.23 [makeQsl2], page 479; Section 7.5.10.22 [makeQso3], page 478; Section 7.5.10.2 [makeUsl], page 465.

7.5.10.25 Qso3Casimir

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: Qso3Casimir(n [,m]), n an integer, m an optional integer

Return: list (of polynomials)

Purpose: compute the Casimir (central) elements of U_q(so_3) for the quantum parameter specialized at the n-th root of unity; if m!=0 is given, polynomials will be normalized

Assume: the basering must be U_q(so_3)

Example:

```
LIB "ncalg.lib";
def R = makeQso3(5);
setring R;
list C = Qso3Casimir(5);
C;
↦ [1]:
↦    1/5*x5+(1/5Q3-1/5Q2+2/5)*x3+(1/5Q3-1/5Q2+1/5)*x
↦ [2]:
↦    1/5*y5+(1/5Q3-1/5Q2+2/5)*y3+(1/5Q3-1/5Q2+1/5)*y
↦ [3]:
↦    1/5*z5+(1/5Q3-1/5Q2+2/5)*z3+(1/5Q3-1/5Q2+1/5)*z
list Cnorm = Qso3Casimir(5,1);
Cnorm;
↦ [1]:
↦    x5+(Q3-Q2+2)*x3+(Q3-Q2+1)*x
↦ [2]:
↦    y5+(Q3-Q2+2)*y3+(Q3-Q2+1)*y
↦ [3]:
↦    z5+(Q3-Q2+2)*z3+(Q3-Q2+1)*z
```

See also: Section 7.5.10.22 [makeQso3], page 478.

7.5.10.26 GKZsystem

Procedure from library `ncalg.lib` (see Section 7.5.10 [ncalg_lib], page 464).

Usage: GKZsystem(A, sord, alg, [,v]); A intmat, sord, alg string, v intvec

Return: ring

Purpose: define a ring (Weyl algebra) and create a Gelfand-Kapranov-Zelevinsky (GKZ) system of equations in a ring from the following data:
A is an intmat, defining the system,
sord is a string with desired term ordering,
alg is a string, saying which algorithm to use (exactly like in toric_lib),
v is an optional intvec.
In addition, the ideal called GKZid containing actual equations is calculated and exported to the ring.

Note: activate the output ring with the setring command. This procedure is elaborated by Oleksandr Iena

Assume: This procedure uses toric_lib and therefore inherits its input requirements:
possible values for input variable alg are: "ect","pt","blr", "hs", "du".
As for the term ordering, it should be a string sord in Singular format like "lp","dp", etc.
Please consult the toric_lib for allowed orderings and more details.

Example:

```
LIB "ncalg.lib";
// example 3.1.4 from the [SST] without the vector w
intmat A[2][4]=3,2,1,0,0,1,2,3;
print(A);
↦      3     2     1     0
↦      0     1     2     3
def D1 = GKZsystem(A,"lp","ect");
setring D1;
D1;
↦ // coefficients: QQ(b(1), b(2))
↦ // number of vars : 8
↦ //        block   1 : ordering a
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //                  : weights     0    0    0    0
↦ //        block   2 : ordering lp
↦ //                  : names    x(1) x(2) x(3) x(4) d(1) d(2) d(3) d(4)
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    d(1)x(1)=x(1)*d(1)+1
↦ //    d(2)x(2)=x(2)*d(2)+1
↦ //    d(3)x(3)=x(3)*d(3)+1
↦ //    d(4)x(4)=x(4)*d(4)+1
print(GKZid);
↦ 3*x(1)*d(1)+2*x(2)*d(2)+x(3)*d(3)+(-b(1)),
↦ x(2)*d(2)+2*x(3)*d(3)+3*x(4)*d(4)+(-b(2)),
↦ d(2)*d(4)-d(3)^2,
↦ d(1)*d(4)-d(2)*d(3),
↦ d(1)*d(3)-d(2)^2
// now, consider A with the vector w=1,1,1,1
intvec v=1,1,1,1;
def D2 = GKZsystem(A,"lp","blr",v);
setring D2;
```

```
print(GKZid);
↦ 3*x(1)*d(1)+2*x(2)*d(2)+x(3)*d(3)+(-b(1)),
↦ x(2)*d(2)+2*x(3)*d(3)+3*x(4)*d(4)+(-b(2)),
↦ d(2)*d(4)-d(3)^2,
↦ d(1)*d(4)-d(2)*d(3),
↦ d(1)*d(3)-d(2)^2
```

See also: Section D.4.35 [toric_lib], page 1387.

7.5.11 ncdecomp_lib

Library: ncdecomp.lib

Purpose: Decomposition of a module into its central characters

Authors: Viktor Levandovskyy, levandov@mathematik.uni-kl.de.

Overview:

This library presents algorithms for the central character decomposition of a module, i.e. a decomposition into generalized weight modules with respect to the center. Based on ideas of O. Khomenko and V. Levandovskyy (see the article [L2] in the References for details).

Procedures:

7.5.11.1 CentralQuot

Procedure from library `ncdecomp.lib` (see Section 7.5.11 [ncdecomp_lib], page 483).

Usage: CentralQuot(M, G), M a module, G an ideal

Assume: G is an ideal in the center of the base ring

Return: module

Purpose: compute the central quotient M:G

Theory: for an ideal G of the center of an algebra and a submodule M of A^n,
the central quotient of M by G is defined to be
M:G := { v in A^n | z*v in M, for all z in G }.

Note: the output module is not necessarily given in a Groebner basis

Example:

```
LIB "ncdecomp.lib";
option(returnSB);
def a = makeUsl2();
setring a;
ideal I = e3,f3,h3-4*h;
I = std(I);
poly C=4*e*f+h^2-2*h;  // C in Z(U(sl2)), the central element
ideal G = (C-8)*(C-24);  // G normal factor in Z(U(sl2)) as an ideal in the center
ideal R = CentralQuot(I,G);  // same as I:G
R;
↦ R[1]=h
↦ R[2]=f
↦ R[3]=e
```

See also: Section 7.5.11.3 [CenCharDec], page 484; Section 7.5.11.2 [CentralSaturation], page 484.

7.5.11.2 CentralSaturation

Procedure from library `ncdecomp.lib` (see Section 7.5.11 [ncdecomp_lib], page 483).

Usage: CentralSaturation(M, T), for a module M and an ideal T

Assume: T is an ideal in the center of the base ring

Return: module

Purpose: compute the central saturation of M by T, that is M:T^{\infty}, by repititive application of `CentralQuot`

Note: the output module is not necessarily a Groebner basis

Example:

```
LIB "ncdecomp.lib";
option(returnSB);
def a = makeUsl2();
setring a;
ideal I = e3,f3,h3-4*h;
I = std(I);
poly C=4*e*f+h^2-2*h;
ideal G = C*(C-8);
ideal R = CentralSaturation(I,G);
R=std(R);
vdim(R);
↦ 5
R;
↦ R[1]=h
↦ R[2]=ef-6
↦ R[3]=f3
↦ R[4]=e3
```

See also: Section 7.5.11.3 [CenCharDec], page 484; Section 7.5.11.1 [CentralQuot], page 483.

7.5.11.3 CenCharDec

Procedure from library `ncdecomp.lib` (see Section 7.5.11 [ncdecomp_lib], page 483).

Usage: CenCharDec(I, C); I a module, C an ideal

Assume: C consists of generators of the center of the base ring

Return: a list L, where each entry consists of three records (if a finite decomposition exists)
L[*][1] ('ideal' type), the central character as a maximal ideal in the center,
L[*][2] ('module' type), the Groebner basis of the weight module, corresponding to the character in L[*][1],
L[*][3] ('int' type) is the vector space dimension of the weight module (-1 in case of infinite dimension);

Purpose: compute a finite decomposition of C into central characters or determine that there is no finite decomposition

Note: actual decomposition is the sum of L[i][2] above;
some modules have no finite decomposition (in such case one gets warning message)
The function `central` in `central.lib` may be used to obtain C, when needed.

Example:

```
LIB "ncdecomp.lib";
printlevel=0;
option(returnSB);
def a = makeUsl2(); // U(sl_2) in characteristic 0
setring a;
ideal I = e3,f3,h3-4*h;
I = twostd(I);           // two-sided ideal generated by I
vdim(I);                 // it is finite-dimensional
↦ 10
ideal Cn = 4*e*f+h^2-2*h; // the only central element
list T = CenCharDec(I,Cn);
T;
↦ [1]:
↦    [1]:
↦       _[1]=4ef+h2-2h-8
↦    [2]:
↦       _[1]=h
↦       _[2]=f
↦       _[3]=e
↦    [3]:
↦       1
↦ [2]:
↦    [1]:
↦       _[1]=4ef+h2-2h
↦    [2]:
↦       _[1]=4ef+h2-2h-8
↦       _[2]=h3-4h
↦       _[3]=fh2-2fh
↦       _[4]=eh2+2eh
↦       _[5]=f2h-2f2
↦       _[6]=e2h+2e2
↦       _[7]=f3
↦       _[8]=e3
↦    [3]:
↦       9
// consider another example
ideal J = e*f*h;
CenCharDec(J,Cn);
↦ There is no finite decomposition
↦ 0
```

See also: Section 7.5.11.1 [CentralQuot], page 483; Section 7.5.11.2 [CentralSaturation], page 484.

7.5.11.4 IntersectWithSub

Procedure from library `ncdecomp.lib` (see Section 7.5.11 [ncdecomp_lib], page 483).

Usage: IntersectWithSub(M,Z), M an ideal, Z an ideal

Assume: Z consists of pairwise commutative elements

Return: ideal of two-sided generators, not a Groebner basis

Purpose: computes the intersection of M with the subalgebra, generated by Z

Note: usually Z consists of generators of the center
The function `central` from `central.lib` may be used to obtain the center Z, if needed.

Example:

```
LIB "ncdecomp.lib";
ring R=(0,a),(e,f,h),Dp;
matrix @d[3][3];
@d[1,2]=-h;   @d[1,3]=2e;   @d[2,3]=-2f;
def r = nc_algebra(1,@d); setring r; // parametric U(sl_2)
ideal I = e,h-a;
ideal C;
C[1] = h^2-2*h+4*e*f; // the center of U(sl_2)
ideal X = IntersectWithSub(I,C);
X;
↦ X[1]=4*ef+h2-2*h+(-a2-2a)
ideal G = e*f, h; // the biggest comm. subalgebra of U(sl_2)
ideal Y = IntersectWithSub(I,G);
Y;
↦ Y[1]=h+(-a)
↦ Y[2]=ef+(-a)
```

7.5.12 ncfactor_lib

Library: ncfactor.lib

Purpose: Tools for factorization in some noncommutative algebras

Authors: Albert Heinle, aheinle at uwaterloo.ca
Viktor Levandovskyy, levandov at math.rwth-aachen.de

Overview: In this library, new methods for factorization on polynomials
are implemented for several types of algebras, namely
- finitely presented (and also free) associative algebras (Letterplace subsystem)
- G-algebras (Plural subsystem), including (q)-Weyl and (q)-shift algebras in 2n variables
The determination of the best algorithm available for users input is done
automatically in the procedure ncfactor().

More detailed description of the algorithms and related publications can be found at @url{https://cs.uwaterloo.ca/~aheinle/}.

Procedures:

7.5.12.1 ncfactor

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: ncfactor(h); h is a polynomial in a non-commutative polynomial algebra over a field k.

Return: list(list)

Purpose: Compute all factorizations of h.

Theory: Implements an ansatz-driven factorization method as outlined by Bell, Heinle and Levandovskyy in "On Noncommutative Finite Factorization Domains".

Assume: - k is a ring, such that factorize can factor any univariate and multivariate commutative polynomial over k.
- There exists at least one variable in the ring.

Note: - works for both PLURAL and LETTERPLACE subsystems
- Every entry of the output list is a list with factors for one possible factorization. The first factor is always a constant (1, if no nontrivial constant could be excluded).

Example:

```
LIB "ncfactor.lib";
// first, an example with PLURAL
def R = makeUsl2();
setring(R);
poly p = e^3*f+e^2*f^2-e^3+e^2*f+2*e*f^2-3*e^2*h-2*e*f*h-8*e^2
+e*f+f^2-4*e*h-2*f*h-7*e+f-h;
ncfactor(p);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       e+1
↦    [3]:
↦       ef-e+f-2h-3
↦    [4]:
↦       e+f
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦       e2f+ef2-e2+f2-2eh-3e-f-2h
↦    [3]:
↦       e+1
kill R;
// an example with LETTERPLACE
LIB "freegb.lib";
↦ // ** redefining tstfreegb (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining tstfreegb (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining setLetterplaceAttributes (LIB "freegb.lib";) ./examples/\
   ncfactor.sing:10
↦ // ** redefining setLetterplaceAttributes (LIB "freegb.lib";) ./examples/\
   ncfactor.sing:10
↦ // ** redefining lst2str (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining mod2str (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining vct2str (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining isVar (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining letplaceGBasis (LIB "freegb.lib";) ./examples/ncfactor.s\
   ing:10
↦ // ** redefining letplaceGBasis (LIB "freegb.lib";) ./examples/ncfactor.s\
   ing:10
↦ // ** redefining lieBracket (LIB "freegb.lib";) ./examples/ncfactor.sing:\
   10
↦ // ** redefining lieBracket (LIB "freegb.lib";) ./examples/ncfactor.sing:\
   10
↦ // ** redefining lpPrint (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpPrint (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining freeGBasis (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining freeGBasis (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining crs (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining polylen (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpDegBound (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining lpDegBound (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining lpVarBlockSize (LIB "freegb.lib";) ./examples/ncfactor.s\
  ing:10
↦ // ** redefining lpVarBlockSize (LIB "freegb.lib";) ./examples/ncfactor.s\
  ing:10
↦ // ** redefining isFreeAlgebra (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining isFreeAlgebra (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining lpNcgenCount (LIB "freegb.lib";) ./examples/ncfactor.sin\
  g:10
↦ // ** redefining lpNcgenCount (LIB "freegb.lib";) ./examples/ncfactor.sin\
  g:10
↦ // ** redefining makeLetterplaceRing (LIB "freegb.lib";) ./examples/ncfac\
  tor.sing:10
↦ // ** redefining makeLetterplaceRing (LIB "freegb.lib";) ./examples/ncfac\
  tor.sing:10
↦ // ** redefining makeLetterplaceRing1 (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining makeLetterplaceRing2 (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining makeLetterplaceRing4 (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining makeLetterplaceRing3 (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining freegbold (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining stringpoly2lplace (LIB "freegb.lib";) ./examples/ncfacto\
  r.sing:10
↦ // ** redefining addplaces (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining sent2lplace (LIB "freegb.lib";) ./examples/ncfactor.sing\
  :10
↦ // ** redefining testnumber (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining str2lplace (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining strpower2rep (LIB "freegb.lib";) ./examples/ncfactor.sin\
  g:10
↦ // ** redefining shiftPoly (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining lastBlock (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining test_shift (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining lp2lstr (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lp2lstr (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining strList2poly (LIB "freegb.lib";) ./examples/ncfactor.sin\
  g:10
↦ // ** redefining file2lplace (LIB "freegb.lib";) ./examples/ncfactor.sing\
  :10
↦ // ** redefining lpPower (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpNF (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpNF (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpDivision (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining lpDivision (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining lpGBPres2Poly (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining lpGBPres2Poly (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining getExpVecs (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining delSupZero (LIB "freegb.lib";) ./examples/ncfactor.sing:\
  10
↦ // ** redefining delSupZeroList (LIB "freegb.lib";) ./examples/ncfactor.s\
  ing:10
↦ // ** redefining makeDVec (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining makeDVecL (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining makeDVecI (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining dShiftDiv (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
  0
↦ // ** redefining lpNormalForm1 (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining lpNormalForm2 (LIB "freegb.lib";) ./examples/ncfactor.si\
  ng:10
↦ // ** redefining isOrderingShiftInvariant (LIB "freegb.lib";) ./examples/\
  ncfactor.sing:10
↦ // ** redefining isOrderingShiftInvariant (LIB "freegb.lib";) ./examples/\
  ncfactor.sing:10
↦ // ** redefining lpMonomialsWithHoles (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining getlpCoeffs (LIB "freegb.lib";) ./examples/ncfactor.sing\
  :10
↦ // ** redefining lpReduce (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining entryViolation (LIB "freegb.lib";) ./examples/ncfactor.s\
  ing:10
↦ // ** redefining checkAssumptionsLPIV (LIB "freegb.lib";) ./examples/ncfa\
  ctor.sing:10
↦ // ** redefining checkAssumptions (LIB "freegb.lib";) ./examples/ncfactor\
  .sing:10
↦ // ** redefining checkLPRing (LIB "freegb.lib";) ./examples/ncfactor.sing\
  :10
↦ // ** redefining checkAssumptionIdeal (LIB "freegb.lib";) ./examples/ncfa\
   ctor.sing:10
↦ // ** redefining checkAssumptionPoly (LIB "freegb.lib";) ./examples/ncfac\
   tor.sing:10
↦ // ** redefining isContainedInVp (LIB "freegb.lib";) ./examples/ncfactor.\
   sing:10
↦ // ** redefining extractLinearPart (LIB "freegb.lib";) ./examples/ncfacto\
   r.sing:10
↦ // ** redefining isLinearVector (LIB "freegb.lib";) ./examples/ncfactor.s\
   ing:10
↦ // ** redefining lpAssumeViolation (LIB "freegb.lib";) ./examples/ncfacto\
   r.sing:10
↦ // ** redefining skip0 (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining ivL2lpI (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining ivL2lpI (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining iv2lp (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining iv2lp (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining iv2lpList (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining iv2lpList (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining iv2lpMat (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining iv2lpMat (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lpId2ivLi (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining lpId2ivLi (LIB "freegb.lib";) ./examples/ncfactor.sing:1\
   0
↦ // ** redefining lp2iv (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lp2iv (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lp2ivId (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining lp2ivId (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining testLift (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining testLift (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining testSyz (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining testSyz (LIB "freegb.lib";) ./examples/ncfactor.sing:10
↦ // ** redefining mod_init (LIB "freegb.lib";) ./examples/ncfactor.sing:10
ring r = 0,(x,y),Dp;
def R = freeAlgebra(r,5); setring(R);
poly p = x*y*x - x;
ncfactor(p);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       x*y-1
↦    [3]:
↦       x
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦       x
↦    [3]:
↦       y*x-1
```

See also: Section 7.5.12.5 [facSubWeyl], page 493; Section 7.5.12.2 [facWeyl], page 491; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.2 facWeyl

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: facWeyl(h); h a polynomial in the nth Weyl algebra

Return: list

Purpose: compute all factorizations of a polynomial in the first Weyl algebra

Theory: Implements the new algorithm by A. Heinle and V. Levandovskyy, see the thesis of A. Heinle

Assume: basering is the nth Weyl algebra, where n in NN.

Note: Every entry of the output list is a list with factors for one possible factorization. The first factor is always a constant (1, if no nontrivial constant could be excluded).

Example:

```
LIB "ncfactor.lib";
ring R = 0,(x1,x2,d1,d2),dp;
matrix C[4][4] = 1,1,1,1,
1,1,1,1,
1,1,1,1,
1,1,1,1;
matrix D[4][4] = 0,0,1,0,
0,0,0,1,
-1,0,0,0,
0,-1,0,0;
def r = nc_algebra(C,D);
setring(r);
poly h = (d1+1)^2*(d1 + x1*d2);
facWeyl(h);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       d1+1
↦    [3]:
↦       d1+1
↦    [4]:
↦       x1*d2+d1
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦       x1*d1*d2+d1^2+x1*d2+d1+2*d2
↦    [3]:
↦       d1+1
```

See also: Section 7.5.12.7 [facFirstShift], page 495; Section 7.5.12.3 [facFirstWeyl], page 492; Section 7.5.12.5 [facSubWeyl], page 493; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.3 facFirstWeyl

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: facFirstWeyl(h); h a polynomial in the first Weyl algebra

Return: list

Purpose: compute all factorizations of a polynomial in the first Weyl algebra

Theory: This function is a wrapper for facWeyl. It exists to make this library downward-compatible with older versions.

Assume: basering is the first Weyl algebra

Note: Every entry of the output list is a list with factors for one possible factorization. The first factor is always a constant (1, if no nontrivial constant could be excluded).

Example:

```
LIB "ncfactor.lib";
ring R = 0,(x,y),dp;
def r = nc_algebra(1,1);
setring(r);
poly h = (x^2*y^2+x)*(x+1);
facFirstWeyl(h);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       x
↦    [3]:
↦       xy2+1
↦    [4]:
↦       x+1
```

See also: Section 7.5.12.6 [facShift], page 494; Section 7.5.12.5 [facSubWeyl], page 493; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.4 testNCfac

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: testNCfac(l[,p,b]); l is a list, p is an optional poly, b is 1 or 0

Return: Case 1: No optional argument. In this case the output is 1, if the entries in the given list represent the same polynomial or 0 otherwise.
Case 2: One optional argument p is given. In this case it returns 1, if all the entries in l are factorizations of p, otherwise 0. Case 3: Second optional b is given. In this case a list is returned containing the difference between the product of each entry in l and p.

Assume: basering is the first Weyl algebra, the entries of l are polynomials

Purpose: Checks whether a list of factorizations contains factorizations of the same element in the first Weyl algebra

Theory: `testNCfac` multiplies out each factorization and checks whether each factorization was a factorization of the same element.
- if there is only a list given, the output will be 0, if it does not contain factorizations of the same element. Otherwise the output will be 1.

- if there is a polynomial in the second argument, then the procedure checks whether the given list contains factorizations of this polynomial. If it does, then the output depends on the third argument. If it is not given, the procedure will check whether the factorizations in the list l are associated to this polynomial and return either 1 or 0, respectively. If the third argument is given, the output will be a list with the length of the given one and in each entry is the product of one entry in l subtracted by the polynomial.

Example:

```
LIB "ncfactor.lib";
ring r = 0,(x,y),dp;
def R = nc_algebra(1,1);
setring R;
poly h = (x^2*y^2+1)*(x^2);
def t1 = facFirstWeyl(h);
//fist a correct list
testNCfac(t1);
↦ 1
//now a correct list with the factorized polynomial
testNCfac(t1,h);
↦ 1
//now we put in an incorrect list without a polynomial
t1[3][3] = y;
testNCfac(t1);
↦ 0
// take h as additional input
testNCfac(t1,h);
↦ 0
// take h as additional input and output list of differences
testNCfac(t1,h,1);
↦ [1]:
↦ 0
↦ [2]:
↦    0
↦ [3]:
↦    -x4y2+x3y3-4x3y+3x2y2-3x2+xy+1
```

See also: Section 7.5.12.7 [facFirstShift], page 495; Section 7.5.12.3 [facFirstWeyl], page 492; Section 7.5.12.5 [facSubWeyl], page 493.

7.5.12.5 facSubWeyl

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: facSubWeyl(h,[x_1,...,x_n,d_1,...,d_n]); h is a polynomial, x_i, d_i are variables in the basering for i in {1,...,n}

Return: list(list)

Assume: x_i, d_i are variables of a basering with d_i*x_i = x_i*d_i+1 for i in {1,...,n}.
That is, they generate the copy of the first Weyl algebra in a basering.
Moreover, h is a polynomial in the x_i,d_i only.
If the list of variables is ommitted, this function will try to figure out itself if h is in a subalgebra that resembles the Weyl algebra.
This function produces an error if the conditions on the variables do not line up or if the

variables contained in h do not belong to a subalgebra of the basering that resembles the Weyl algebra.

Purpose: compute factorizations of the polynomial, depending on x_i and d_i.

Example:

```
LIB "ncfactor.lib";
ring r = 0,(x,y,z),dp;
matrix D[3][3]; D[1,3]=-1;
def R = nc_algebra(1,D); // x,z generate Weyl subalgebra
setring R;
poly h = (x^2*z^2+x)*x;
list fact1 = facSubWeyl(h,x,z);
// compare with facFirstWeyl:
ring s = 0,(z,x),dp;
def S = nc_algebra(1,1); setring S;
poly h = (x^2*z^2+x)*x;
list fact2 = facFirstWeyl(h);
map F = R,x,0,z;
list fact1 = F(fact1); // it is identical to list fact2
testNCfac(fact1); // check the correctness again
↦ 1
```

See also: Section 7.5.12.7 [facFirstShift], page 495; Section 7.5.12.3 [facFirstWeyl], page 492; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.6 facShift

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: facShift(h); h a polynomial in the n'th shift algebra

Return: list

Purpose: compute all factorizations of a polynomial in the nth shift algebra

Theory: Currently, we do not have a specialized algorithm for the shift algebra in this library that takes advantage of the graded structure, hence this function is mapping to the general factorization algorithm for G-Algebras

Note: Every entry of the output list is a list with factors for one possible factorization.

Example:

```
LIB "ncfactor.lib";
ring R = 0,(x1,x2,s1,s2),dp;
matrix C[4][4] = 1,1,1,1,
1,1,1,1,
1,1,1,1,
1,1,1,1;
matrix D[4][4] = 0,0,s1,0,
0,0,0,s2,
-s1,0,0,0,
0,-s2,0,0;
def r = nc_algebra(C,D);
setring(r);
poly h = x1*(x1+1)*s1^2-2*x1*(x1+100)*s1+(x1+99)*(x1+100);
facShift(h);
```

```
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       x1*s1-x1+s1-100
↦    [3]:
↦       x1*s1-x1-s1-99
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦       x1*s1-x1-100
↦    [3]:
↦       x1*s1-x1-99
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦       x1*s1-x1-99
↦    [3]:
↦       x1*s1-x1-100
```

See also: Section 7.5.12.3 [facFirstWeyl], page 492; Section 7.5.12.5 [facSubWeyl], page 493; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.7 facFirstShift

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: facFirstShift(h); h a polynomial in the first shift algebra

Return: list

Purpose: compute all factorizations of a polynomial in the first shift algebra

Theory: This function is a wrapper for facShift. It exists to make this library downward-compatible with older versions.

Assume: basering is the first shift algebra

Note: Every entry of the output list is a list with factors for one possible factorization.

Example:

```
LIB "ncfactor.lib";
ring R = 0,(x,s),dp;
def r = nc_algebra(1,s);
setring(r);
poly h = (s^2*x+x)*s;
facFirstShift(h);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       s
↦    [3]:
↦       s2+1
↦    [4]:
```

```
↦          x-1
↦    [2]:
↦       [1]:
↦    1
↦       [2]:
↦          s2+1
↦       [3]:
↦          s
↦       [4]:
↦          x-1
↦    [3]:
↦       [1]:
↦    1
↦       [2]:
↦          s2+1
↦       [3]:
↦          x
↦       [4]:
↦          s
```

See also: Section 7.5.12.3 [facFirstWeyl], page 492; Section 7.5.12.5 [facSubWeyl], page 493; Section 7.5.12.4 [testNCfac], page 492.

7.5.12.8 homogfacNthWeyl

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: homogfacNthWeyl(h); h is a homogeneous polynomial in the nth Weyl algebra with respect to the -1,1-grading

Return: list

Purpose: Computes a factorization of a homogeneous polynomial h with respect to the ZZ-grading on the n-th Weyl algebra.

Theory: `homogfacFirstWeyl` returns a list with a factorization of the given, [-1,1]-homogeneous polynomial. For every i in 1..n: If the degree of the polynomial in [d_i,x_i] is k with k positive, the last k entries in the output list are the second variable. If k is positive, the last k entries will be x_i. The other entries will be irreducible polynomials of degree zero or 1 resp. -1. resp. other variables

General assumptions:
- The basering is the nth Weyl algebra and has the form, that the first n variables represent x1, ..., xn, and the second n variables do represent the d1, ..., dn.

7.5.12.9 homogfacNthQWeyl

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: homogfacNthQWeyl(h); h is a homogeneous polynomial in the n'th q-Weyl algebra with respect to the weight vector
`@` [-1,...,-1,1,...,1].
`@` __ __/ __ __/ `@` \/ \/ `@` n/2 n/2

Return: list

Purpose: Computes a factorization of a homogeneous polynomial h in the n'th q-Weyl algebra

Theory: homogfacNthQWeyl returns a list with a factorization of the given, [-1,1]-homogeneous polynomial. For every i in 1..n: If the degree of the polynomial in [d_i,x_i] is k with k positive, the last entries in the output list are the second variable. If k is positive, the last k entries will be x_i. The other entries will be irreducible polynomials of degree zero or 1 resp. -1. resp. other variables

General assumptions:

- The basering is the nth Weyl algebra and has the form, that the first n variables represent x1, ..., xn, and the second n variables do represent the d1, ..., dn.
- We have n parameters q_1,..., q_n given.

Example:

```
LIB "ncfactor.lib";
ring R = (0,q1,q2,q3),(x1,x2,x3,d1,d2,d3),dp;
matrix C[6][6] = 1,1,1,q1,1,1,
1,1,1,1,q2,1,
1,1,1,1,1,q3,
1,1,1,1,1,1,
1,1,1,1,1,1,
1,1,1,1,1,1;
matrix D[6][6] = 0,0,0,1,0,0,
0,0,0,0,1,0,
0,0,0,0,0,1,
-1,0,0,0,0,0,
0,-1,0,0,0,0,
0,0,-1,0,0,0;
def r = nc_algebra(C,D);
setring(r);
poly h =x1*x2^2*x3^3*d1*d2^2+x2*x3^3*d2;
homogfacNthQWeyl(h);
↦ [1]:
↦    1/(q2)
↦ [2]:
↦    x2
↦ [3]:
↦    d2
↦ [4]:
↦    x1*x2*d1*d2-x1*d1+(q2)
↦ [5]:
↦    x3
↦ [6]:
↦    x3
↦ [7]:
↦    x3
```

See also: Section 7.5.12.10 [homogfacFirstQWeyl], page 497; Section 7.5.12.12 [homogfacFirstQWeyl_all], page 516; Section 7.5.12.11 [homogfacNthQWeyl_all], page 498.

7.5.12.10 homogfacFirstQWeyl

Procedure from library ncfactor.lib (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: homogfacFirstQWeyl(h); h is a homogeneous polynomial in the first q-Weyl algebra with respect to the weight vector [-1,1]

Return: list

Purpose: Computes a factorization of a homogeneous polynomial h with respect to the weight vector [-1,1] in the first q-Weyl algebra

Theory: This function is a wrapper for homogfacNthQWeyl. It exists to make this library downward-compatible with older versions.

Example:

```
LIB "ncfactor.lib";
ring R = (0,q),(x,d),dp;
def r = nc_algebra (q,1);
setring(r);
poly h = q^25*x^10*d^10+q^16*(q^4+q^3+q^2+q+1)^2*x^9*d^9+
q^9*(q^13+3*q^12+7*q^11+13*q^10+20*q^9+26*q^8+30*q^7+
31*q^6+26*q^5+20*q^4+13*q^3+7*q^2+3*q+1)*x^8*d^8+
q^4*(q^9+2*q^8+4*q^7+6*q^6+7*q^5+8*q^4+6*q^3+
4*q^2+2q+1)*(q^4+q^3+q^2+q+1)*(q^2+q+1)*x^7*d^7+
q*(q^2+q+1)*(q^5+2*q^4+2*q^3+3*q^2+2*q+1)*(q^4+q^3+q^2+q+1)*(q^2+1)*(q+1)*x^6*d^6+
(q^10+5*q^9+12*q^8+21*q^7+29*q^6+33*q^5+31*q^4+24*q^3+15*q^2+7*q+12)*x^5*d^5+
6*x^3*d^3+24;
homogfacFirstQWeyl(h);
↦ [1]:
↦    1
↦ [2]:
↦    x5d5+6
↦ [3]:
↦    x5d5+x3d3+4
```

See also: Section 7.5.12.12 [homogfacFirstQWeyl_all], page 516.

7.5.12.11 homogfacNthQWeyl_all

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: homogfacNthQWeyl_all(h); h is a homogeneous polynomial in the n'th q-Weyl algebra with respect to the weight vector
@ [-1,...,-1,1,...,1].
@ \-- --/ \-- --/ @ \/ \/ @ n/2 n/2

Return: list

Purpose: Computes all factorizations of a homogeneous polynomial h in the n'th q-Weyl algebra

Theory: `homogfacNthQWeyl` returns a list with lists representing each a factorization of the given,
[-1,...,-1,1,...,1]-homogeneous polynomial.

General assumptions:
- The basering is the nth Weyl algebra and has the form, that the first n variables represent x1, ..., xn, and the second n variables do represent the d1, ..., dn. - We have n parameters q_1,..., q_n given.

Example:

```
LIB "ncfactor.lib";
ring R = (0,q1,q2,q3),(x1,x2,x3,d1,d2,d3),dp;
matrix C[6][6] = 1,1,1,q1,1,1,
1,1,1,1,q2,1,
1,1,1,1,1,q3,
1,1,1,1,1,1,
1,1,1,1,1,1,
1,1,1,1,1,1;
matrix D[6][6] = 0,0,0,1,0,0,
0,0,0,0,1,0,
0,0,0,0,0,1,
-1,0,0,0,0,0,
0,-1,0,0,0,0,
0,0,-1,0,0,0;
def r = nc_algebra(C,D);
setring(r);
poly h =x1*x2^2*x3^3*d1*d2^2+x2*x3^3*d2;
homogfacNthQWeyl_all(h);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       x2
↦    [3]:
↦       x1*x2*d1*d2+1
↦    [4]:
↦       d2
↦    [5]:
↦       x3
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦       x2
↦    [3]:
↦       x1*x2*d1*d2+1
↦    [4]:
↦       x3
↦    [5]:
↦       d2
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦       x2
↦    [3]:
↦       x1*x2*d1*d2+1
↦    [4]:
```

```
↦          x3
↦       [5]:
↦          x3
↦       [6]:
↦          d2
↦       [7]:
↦          x3
↦    [4]:
↦       [1]:
↦ 1
↦       [2]:
↦          x2
↦       [3]:
↦          x1*x2*d1*d2+1
↦       [4]:
↦          x3
↦       [5]:
↦          x3
↦       [6]:
↦          x3
↦       [7]:
↦          d2
↦    [5]:
↦       [1]:
↦ 1
↦       [2]:
↦          x2
↦       [3]:
↦          x3
↦       [4]:
↦          x1*x2*d1*d2+1
↦       [5]:
↦          d2
↦       [6]:
↦          x3
↦       [7]:
↦          x3
↦    [6]:
↦       [1]:
↦ 1
↦       [2]:
↦          x2
↦       [3]:
↦          x3
↦       [4]:
↦          x1*x2*d1*d2+1
↦       [5]:
↦          x3
↦       [6]:
↦          d2
↦       [7]:
↦          x3
↦    [7]:
```

```
↦     [1]:
↦ 1
↦     [2]:
↦        x2
↦     [3]:
↦        x3
↦     [4]:
↦        x1*x2*d1*d2+1
↦     [5]:
↦        x3
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦ [8]:
↦     [1]:
↦ 1
↦     [2]:
↦        x2
↦     [3]:
↦        x3
↦     [4]:
↦        x3
↦     [5]:
↦        x1*x2*d1*d2+1
↦     [6]:
↦        d2
↦     [7]:
↦        x3
↦ [9]:
↦     [1]:
↦ 1
↦     [2]:
↦        x2
↦     [3]:
↦        x3
↦     [4]:
↦        x3
↦     [5]:
↦        x1*x2*d1*d2+1
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦ [10]:
↦     [1]:
↦ 1
↦     [2]:
↦        x2
↦     [3]:
↦        x3
↦     [4]:
↦        x3
↦     [5]:
↦        x3
↦     [6]:
↦        x1*x2*d1*d2+1
↦     [7]:
↦        d2
↦  [11]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x2
↦     [4]:
↦        x1*x2*d1*d2+1
↦     [5]:
↦        d2
↦     [6]:
↦        x3
↦     [7]:
↦        x3
↦  [12]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x2
↦     [4]:
↦        x1*x2*d1*d2+1
↦     [5]:
↦        x3
↦     [6]:
↦        d2
↦     [7]:
↦        x3
↦  [13]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x2
↦     [4]:
↦        x1*x2*d1*d2+1
↦     [5]:
↦        x3
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦  [14]:
↦     [1]:
```

```
↦ 1
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       x1*x2*d1*d2+1
↦    [6]:
↦       d2
↦    [7]:
↦       x3
↦ [15]:
↦    [1]:
↦ 1
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       x1*x2*d1*d2+1
↦    [6]:
↦       x3
↦    [7]:
↦       d2
↦ [16]:
↦    [1]:
↦ 1
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       x1*x2*d1*d2+1
↦    [7]:
↦       d2
↦ [17]:
↦    [1]:
↦ 1
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
↦       x2
↦    [5]:
```

```
↦        x1*x2*d1*d2+1
↦     [6]:
↦        d2
↦     [7]:
↦        x3
↦  [18]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        x1*x2*d1*d2+1
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦  [19]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        x3
↦     [6]:
↦        x1*x2*d1*d2+1
↦     [7]:
↦        d2
↦  [20]:
↦     [1]:
↦  1
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x3
↦     [5]:
↦        x2
↦     [6]:
↦        x1*x2*d1*d2+1
↦     [7]:
↦        d2
↦  [21]:
↦     [1]:
↦  1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x2
↦      [4]:
↦         d2
↦      [5]:
↦         x3
↦      [6]:
↦         x3
↦      [7]:
↦         x3
↦   [22]:
↦      [1]:
↦   1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x2
↦      [4]:
↦         x3
↦      [5]:
↦         d2
↦      [6]:
↦         x3
↦      [7]:
↦         x3
↦   [23]:
↦      [1]:
↦   1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x2
↦      [4]:
↦         x3
↦      [5]:
↦         x3
↦      [6]:
↦         d2
↦      [7]:
↦         x3
↦   [24]:
↦      [1]:
↦   1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x2
↦      [4]:
↦         x3
↦      [5]:
↦         x3
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦  [25]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        d2
↦     [6]:
↦        x3
↦     [7]:
↦        x3
↦  [26]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        x3
↦     [6]:
↦        d2
↦     [7]:
↦        x3
↦  [27]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        x3
↦     [6]:
↦        x3
↦     [7]:
↦        d2
↦  [28]:
↦     [1]:
↦  1/(q2)
↦     [2]:
```

```
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x3
↦      [4]:
↦         x3
↦      [5]:
↦         x2
↦      [6]:
↦         d2
↦      [7]:
↦         x3
↦  [29]:
↦      [1]:
↦  1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x3
↦      [4]:
↦         x3
↦      [5]:
↦         x2
↦      [6]:
↦         x3
↦      [7]:
↦         d2
↦  [30]:
↦      [1]:
↦  1/(q2)
↦      [2]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [3]:
↦         x3
↦      [4]:
↦         x3
↦      [5]:
↦         x3
↦      [6]:
↦         x2
↦      [7]:
↦         d2
↦  [31]:
↦      [1]:
↦  1/(q2)
↦      [2]:
↦         x2
↦      [3]:
↦         d2
↦      [4]:
↦         x1*x2*d1*d2-x1*d1+(q2)
↦      [5]:
↦         x3
↦      [6]:
```

```
↦       x3
↦    [7]:
↦       x3
↦ [32]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       d2
↦    [4]:
↦       x3
↦    [5]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [33]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       d2
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [7]:
↦       x3
↦ [34]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       d2
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       x3
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [35]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       d2
↦    [5]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [36]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       d2
↦    [5]:
↦       x3
↦    [6]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [7]:
↦       x3
↦ [37]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       d2
↦    [5]:
↦       x3
↦    [6]:
↦       x3
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [38]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       x3
↦    [5]:
↦       d2
↦    [6]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [7]:
↦       x3
↦ [39]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       x3
↦    [5]:
↦       d2
↦    [6]:
↦       x3
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [40]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x2
↦    [3]:
↦       x3
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       d2
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [41]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [4]:
↦       x2
↦    [5]:
↦       d2
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [42]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
```

```
↦          x1*x2*d1*d2-x1*d1+(q2)
↦       [4]:
↦          x2
↦       [5]:
↦          x3
↦       [6]:
↦          d2
↦       [7]:
↦          x3
↦    [43]:
↦       [1]:
↦    1/(q2)
↦       [2]:
↦          x3
↦       [3]:
↦          x1*x2*d1*d2-x1*d1+(q2)
↦       [4]:
↦          x2
↦       [5]:
↦          x3
↦       [6]:
↦          x3
↦       [7]:
↦          d2
↦    [44]:
↦       [1]:
↦    1/(q2)
↦       [2]:
↦          x3
↦       [3]:
↦          x1*x2*d1*d2-x1*d1+(q2)
↦       [4]:
↦          x3
↦       [5]:
↦          x2
↦       [6]:
↦          d2
↦       [7]:
↦          x3
↦    [45]:
↦       [1]:
↦    1/(q2)
↦       [2]:
↦          x3
↦       [3]:
↦          x1*x2*d1*d2-x1*d1+(q2)
↦       [4]:
↦          x3
↦       [5]:
↦          x2
↦       [6]:
↦          x3
↦       [7]:
```

```
↦       d2
↦ [46]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       x2
↦    [7]:
↦       d2
↦ [47]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       d2
↦    [5]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [6]:
↦       x3
↦    [7]:
↦       x3
↦ [48]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       d2
↦    [5]:
↦       x3
↦    [6]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [7]:
↦       x3
↦ [49]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       d2
↦    [5]:
↦       x3
↦    [6]:
↦       x3
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [50]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       d2
↦    [6]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [7]:
↦       x3
↦ [51]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       d2
↦    [6]:
↦       x3
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [52]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x2
↦    [4]:
↦       x3
↦    [5]:
↦       x3
↦    [6]:
↦       d2
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦ [53]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [5]:
↦       x2
↦    [6]:
↦       d2
↦    [7]:
↦       x3
↦ [54]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [5]:
↦       x2
↦    [6]:
↦       x3
↦    [7]:
↦       d2
↦ [55]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
↦       x1*x2*d1*d2-x1*d1+(q2)
↦    [5]:
↦       x3
↦    [6]:
↦       x2
↦    [7]:
↦       d2
↦ [56]:
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
```

```
↦        x2
↦     [5]:
↦        d2
↦     [6]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦     [7]:
↦        x3
↦  [57]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        d2
↦     [6]:
↦        x3
↦     [7]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦  [58]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x2
↦     [5]:
↦        x3
↦     [6]:
↦        d2
↦     [7]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦  [59]:
↦     [1]:
↦  1/(q2)
↦     [2]:
↦        x3
↦     [3]:
↦        x3
↦     [4]:
↦        x3
↦     [5]:
↦        x1*x2*d1*d2-x1*d1+(q2)
↦     [6]:
↦        x2
↦     [7]:
↦        d2
↦  [60]:
```

```
↦    [1]:
↦ 1/(q2)
↦    [2]:
↦       x3
↦    [3]:
↦       x3
↦    [4]:
↦       x3
↦    [5]:
↦       x2
↦    [6]:
↦       d2
↦    [7]:
↦       x1*x2*d1*d2-x1*d1+(q2)
```

See also: Section 7.5.12.10 [homogfacFirstQWeyl], page 497; Section 7.5.12.12 [homogfacFirstQWeyl_all], page 516; Section 7.5.12.8 [homogfacNthWeyl], page 496.

7.5.12.12 homogfacFirstQWeyl_all

Procedure from library `ncfactor.lib` (see Section 7.5.12 [ncfactor_lib], page 486).

Usage: homogfacFirstQWeyl_all(h); h is a homogeneous polynomial in the first q-Weyl algebra with respect to the weight vector [-1,1]

Return: list

Purpose: Computes all factorizations of a homogeneous polynomial h with respect to the weight vector [-1,1] in the first q-Weyl algebra

Theory: This function is a wrapper for homogFacNthQWeyl_all. It exists to make this library downward-compatible with older versions.

Example:

```
LIB "ncfactor.lib";
ring R = (0,q),(x,d),dp;
def r = nc_algebra (q,1);
setring(r);
poly h = q^25*x^10*d^10+q^16*(q^4+q^3+q^2+q+1)^2*x^9*d^9+
q^9*(q^13+3*q^12+7*q^11+13*q^10+20*q^9+26*q^8+30*q^7+
31*q^6+26*q^5+20*q^4+13*q^3+7*q^2+3*q+1)*x^8*d^8+
q^4*(q^9+2*q^8+4*q^7+6*q^6+7*q^5+8*q^4+6*q^3+
4*q^2+2q+1)*(q^4+q^3+q^2+q+1)*(q^2+q+1)*x^7*d^7+
q*(q^2+q+1)*(q^5+2*q^4+2*q^3+3*q^2+2*q+1)*(q^4+q^3+q^2+q+1)*(q^2+1)*(q+1)*x^6*d^6+
(q^10+5*q^9+12*q^8+21*q^7+29*q^6+33*q^5+31*q^4+24*q^3+15*q^2+7*q+12)*x^5*d^5+
6*x^3*d^3+24;
homogfacFirstQWeyl_all(h);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦       x5d5+6
↦    [3]:
↦       x5d5+x3d3+4
↦ [2]:
↦    [1]:
```

```
↦ 1
↦    [2]:
↦       x5d5+x3d3+4
↦    [3]:
↦       x5d5+6
```

See also: Section 7.5.12.10 [homogfacFirstQWeyl], page 497.

7.5.13 nchilbert_lib

Library: nchilbert.lib

Purpose: Hilbert series, polynomial and multiplicity for G-Algebras (Plural)

Authors: Andre Ranft, andre.ranft at rwth-aachen.de
Viktor Levandovskyy, levandov at rwth-aachen.de

Overview: The theory is found in the book by Bueso, Gomez-Torrecillas, and Verschoren Algorithmic Methods in Non-Commutative Algebra. Applications to Quantum Groups. and in the bachelor thesis by
Andre Ranft, Hilbert polynomials of modules over noncommutative G-algebras, RWTH Aachen, 2014.

Procedures: See also: Section 5.1.56 [hilb], page 195; Section D.15.11 [multigrading_lib], page 2458; Section D.7.3 [rinvar_lib], page 1806.

7.5.13.1 ncHilb

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: ncHilb(M,j); M a module, j an int

Return: intvec

Assume: - M is given via a Groebner basis; j is 1 or 2;
- the weights of all the ring variables are positive

Note: - computes the first (if j=1) or second (j=2) Hilbert series of I as intvec - the procedure works analogously to the commutative procedure `hilb` - If the returned vector has the form `v=(v_0,v_1,...,v_d,0)`, then the Hilbert series is `v_0 + v_1*t + ... + v_d*t^d`

Example:

```
LIB "nchilbert.lib";
def A = makeUsl2(); setring A;
ideal I = e,h-1; I = std(I);
ncHilb(I,1); // first Hilbert series of A/I
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1,-2,1,0
ncHilb(I,2); // second Hilbert series of A/I
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1,0
ideal J = I, f^2; J = std(J);
ncHilb(J,2);
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1,1,0
// now with weights 1,2,3
ring r = 0,(e,f,h),wp(1,2,3);
matrix D[3][3]; D[1,2]=-h; D[1,3]=2*e;D[2,3]=-2*f;
def R = nc_algebra(1,D); setring R;
ideal I = imap(A,I); I = std(I);
ncHilb(I,1); // first weighted Hilbert series of R/I
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1,-1,0,-1,1,0
ncHilb(I,2); // second weighted Hilbert series of R/I
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1,1,1,0
matrix M[2][5] =
e,h-1,f^2, 0,0,
0,0,0, e,h+1;
module G = std(M);
print(G);
↦ e,0,h-1,0,  f2,
↦ 0,e,0,  h+1,0
ncHilb(G,1); // first weighted Hilbert series of R^2/G
↦ 2,-2,0,-2,1,1,0,1,-1,0
ncHilb(G,2); // second weighted Hilbert series of R^2/G
↦ 2,2,2,0,-1,-1,-1,0
```

See also: Section 5.1.56 [hilb], page 195.

7.5.13.2 ncHilbertSeries

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: ncHilbertSeries(M,j); M is a module, j is an int

Return: ring

Purpose: computes the first (if j=1) and second (j=2) Hilbert Series of A^m/M

Assume: - M is given via a Groebner basis; j is 1 or 2;
- the weights of all the ring variables are positive

Note: - the procedure returns an univariate ring and a polynomial called `ncHS` in it.

Example:

```
LIB "nchilbert.lib";
def A = makeUsl2(); setring A;
ideal I = e,h-1; I = std(I);
def r = ncHilbertSeries(I,1); setring r;
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
ncHS;
↦ t2-2t+1
setring A; kill r;
def s= ncHilbertSeries(I,2); setring s;
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
```

```
ncHS;
↦ 1
// now consider admissible weights 1,2,3
ring r = 0,(e,f,h),wp(1,2,3);
matrix D[3][3]; D[1,2]=-h; D[1,3]=2*e;D[2,3]=-2*f;
def R = nc_algebra(1,D); setring R;
matrix M[2][5] =
e,h-1,f^2, 0,0,
0,0,0, e,h+1;
module G = std(M);
print(G);
↦ e,0,h-1,0,  f2,
↦ 0,e,0,  h+1,0
def r= ncHilbertSeries(G,1);  setring r;
↦ // ** redefining r (def r= ncHilbertSeries(G,1);  setring r;) ./examples/\
   ncHilbertSeries.sing:18
ncHS; // first weighted Hilbert series of R^2/G
↦ -t8+t7+t5+t4-2t3-2t+2
setring R; kill r;
def s=ncHilbertSeries(G,2); setring s;
↦ // ** redefining s (def s=ncHilbertSeries(G,2); setring s;) ./examples/nc\
   HilbertSeries.sing:21
ncHS;// second weighted Hilbert series of R^2/G
↦ -t6-t5-t4+2t2+2t+2
```

7.5.13.3 ncHilbertPolynomial

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: ncHilbertPolynomial(M); M is a module

Return: ring

Purpose: computes the Hilbert polynomial of A^m/M

Assume: - M is given via a Groebner basis
- the weights of all the ring variables are positive

Note: - the procedure returns an univariate ring and a polynomial called `ncHP` in it.

Example:

```
LIB "nchilbert.lib";
def A = makeUsl2(); setring A;
ideal I = h^4,e*f*h^3,e^2*f^2*h^2+2*e*f*h^2; I = std(I);
dim(I); // 2
↦ 2
def r = ncHilbertPolynomial(I); setring r;
ncHP; // 2t+7
↦ 2t+7
kill r;
// now consider admissible weights 1,2,3
ring r = 0,(e,f,h),wp(1,2,3);
matrix D[3][3]; D[1,2]=-h; D[1,3]=2*e;D[2,3]=-2*f;
def R = nc_algebra(1,D); setring R;
ideal I = imap(A,I);
I = std(I);
```

```
dim(I); // 2
↦ 2
def r = ncHilbertPolynomial(I); setring r;
↦ // ** redefining r (def r = ncHilbertPolynomial(I); setring r;) ./example\
   s/ncHilbertPolynomial.sing:15
ncHP; // 6t+18
↦ 6t+18
```

See also: Section D.2.8.20 [hilbPoly], page 897.

7.5.13.4 ncHilbertMultiplicity

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: ncHilbertMultiplicity(M); M is a module

Return: int

Purpose: compute the (Hilbert) multiplicity of the module coker(M)

Assume: - M is given via a Groebner basis
- the weights of all the ring variables are positive

Note: the multiplicity depends on the selected weights of variables

Example:

```
LIB "nchilbert.lib";
def A = makeUsl2(); setring A;
ideal I = e,h-1; I = std(I);
ncHilbertMultiplicity(I); // multiplicity of A/I
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 1
ideal J = I, f^2; J = std(J);
ncHilbertMultiplicity(J);
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 2
// now the same algebra with weights 1,2,3
ring r = 0,(e,f,h),wp(1,2,3);
matrix D[3][3]; D[1,2]=-h; D[1,3]=2*e;D[2,3]=-2*f;
def R = nc_algebra(1,D); setring R;
ideal I = imap(A,I); I = std(I);
ncHilbertMultiplicity(I);
↦ Warning: the input generators are not a Groebner basis
↦ The result might have no meaning
↦ 3
matrix M[2][5] =
e,h-1,f^2, 0,0,
0,0,0, e,h+1;
module G = std(M);
print(G);
↦ e,0,h-1,0,  f2,
↦ 0,e,0,  h+1,0
ncHilbertMultiplicity(G);
↦ 3
```

7.5.13.5 GKExp

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: GKExp(M); M a module

Return: int

Purpose: computes the Gelfand-Kirillov-Dimension of coker(M) via Exp(M)

Assume: basering is G-Algebra

Note: for zero module -1 is returned

Example:

```
LIB "nchilbert.lib";
def A = makeUsl2(); setring A;
ideal I = e,h-1; I = std(I);
GKExp(I); // computes GKdim(A/I), should be 1
↦ Warning: the input generators are not a Groebner basis
↦ Proceed with Groebner basis computation
↦ 1
ideal J = I, f^2; J = std(J);
GKExp(J); // should be 0
↦ Warning: the input generators are not a Groebner basis
↦ Proceed with Groebner basis computation
↦ 0
matrix M[2][4] =
e,h-1,0,0,
0,0,e,h+1;
module G = std(M);
print(G);
↦ h-1,0,  e,0,
↦ 0,  h+1,0,e
GKExp(G);
↦ 1
```

See also: Section 7.5.9.1 [GKdim], page 464; Section 7.3.3 [dim (plural)], page 336.

7.5.13.6 mondim

Procedure from library `nchilbert.lib` (see Section 7.5.13 [nchilbert_lib], page 517).

Usage: mondim(B,i); B is list of elements of N_0^i,

Return: int

Purpose: computes the dimension of the monoid ideal generated by B

Example:

```
LIB "nchilbert.lib";
ring A = 0,(x,y,z),dp;
setring A;
list I = [1,0,1],[0,1,1]; // belongs to the ideal <xz,yz>
mondim(I,3);
↦ 1
mondim(I,5); // treat generators of I as extended in N_0^5
↦ 3
```

7.5.14 dmodloc_lib

Status: experimental

Library: dmodloc.lib

Purpose: Localization of algebraic D-modules and applications

Author: Daniel Andres, daniel.andres@math.rwth-aachen.de
Support: DFG Graduiertenkolleg 1632 'Experimentelle und konstruktive Algebra'

Overview: Let I be a left ideal in the n-th polynomial Weyl algebra D=K[x]<d> and let f be a polynomial in K[x].

If D/I is a holonomic module over D, it is known that the localization of D/I at f is also holonomic. The procedure `Dlocalization` computes an ideal J in D such that this localization is isomorphic to D/J as D-modules.

If one regards I as an ideal in the rational Weyl algebra as above, K(x)<d>*I, and intersects with K[x]<d>, the result is called the Weyl closure of I. The procedures `WeylClosure` (if I has finite holonomic rank) and `WeylClosure1` (if I is in the first Weyl algebra) can be used for computations.

As an application of the Weyl closure, the procedure `annRatSyz` computes a holonomic part of the annihilator of a rational function by computing certain syzygies. The full annihilator can be obtained by taking the Weyl closure of the result.

If one regards the left ideal I as system of linear PDEs, one can find its polynomial solutions with `polSol` (if I is holonomic) or `polSolFiniteRank` (if I is of finite holonomic rank). Rational solutions can be obtained with `ratSol`.

The procedure `bfctBound` computes a possible multiple of the b-function for f^s*u at a generic root of f. Here, u stands for [1] in D/I.

This library also offers the procedures `holonomicRank` and `DsingularLocus` to compute the holonomic rank and the singular locus of the D-module D/I.

References:

(OT) T. Oaku, N. Takayama: 'Algorithms for D-modules', Journal of Pure and Applied Algebra, 1998.
(OTT) T. Oaku, N. Takayama, H. Tsai: 'Polynomial and rational solutions of holonomic systems', Journal of Pure and Applied Algebra, 2001.
(OTW) T. Oaku, N. Takayama, U. Walther: 'A Localization Algorithm for D-modules', Journal of Symbolic Computation, 2000.
(Tsa) H. Tsai: 'Algorithms for algebraic analysis', PhD thesis, 2000.

Procedures: See also: Section 7.5.2 [bfun_lib], page 375; Section 7.5.4 [dmod_lib], page 400; Section 7.5.5 [dmodapp_lib], page 420; Section 7.5.7 [dmodvar_lib], page 453; Section D.6.13 [gmssing_lib], page 1702.

7.5.14.1 Dlocalization

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: Dlocalization(I,f[,k,e]); I ideal, f poly, k,e optional ints

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that f does not contain any D(i) and that I is holonomic on K^n\V(f).

Return: ideal or list, computes an ideal J such that D/J is isomorphic to D/I localized at f as D-modules.
If k<>0, a list consisting of J and an integer m is returned, such that f^m represents the natural map from D/I to D/J. Otherwise (and by default), only the ideal J is returned.

Remarks: It is known that a localization at f of a holonomic D-module is again a holonomic D-module.
Reference: (OTW)

Note: If e<>0, `std` is used for Groebner basis computations, otherwise (and by default) `slimgb` is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
// (OTW), Example 8
ring r = 0,(x,y,z,Dx,Dy,Dz),dp;
def W = Weyl();
setring W;
poly f = x^3-y^2*z^2;
ideal I = f^2*Dx+3*x^2, f^2*Dy-2*y*z^2, f^2*Dz-2*y^2*z;
// I annihilates exp(1/f)
ideal J = Dlocalization(I,f);
J;
↦ J[1]=y*Dy-z*Dz
↦ J[2]=2*y*z^2*Dx+3*x^2*Dy
↦ J[3]=2*y^2*z*Dx+3*x^2*Dz
↦ J[4]=2*z^3*Dx*Dz+3*x^2*Dy^2+2*z^2*Dx
↦ J[5]=3*y^2*z^3*Dz-2*x^4*Dx-6*x^3*z*Dz+12*y^2*z^2-12*x^3-6
↦ J[6]=4*x^4*Dx^2+12*x^3*z*Dx*Dz+9*x^2*z^2*Dz^2+40*x^3*Dx+63*x^2*z*Dz+72*x^\
   2+12*Dx
↦ J[7]=3*y*z^4*Dz^2-2*x^4*Dx*Dy-6*x^3*z*Dy*Dz+21*y*z^3*Dz-12*x^3*Dy+24*y*z^\
   2-6*Dy
↦ J[8]=3*z^5*Dz^3-2*x^4*Dx*Dy^2-6*x^3*z*Dy^2*Dz+30*z^4*Dz^2-12*x^3*Dy^2+66*\
   z^3*Dz+24*z^2-6*Dy^2
Dlocalization(I,f,1); // The natural map D/I -> D/J is given by 1/f^2
↦ [1]:
↦    _[1]=y*Dy-z*Dz
↦    _[2]=2*y*z^2*Dx+3*x^2*Dy
↦    _[3]=2*y^2*z*Dx+3*x^2*Dz
↦    _[4]=2*z^3*Dx*Dz+3*x^2*Dy^2+2*z^2*Dx
↦    _[5]=3*y^2*z^3*Dz-2*x^4*Dx-6*x^3*z*Dz+12*y^2*z^2-12*x^3-6
↦    _[6]=4*x^4*Dx^2+12*x^3*z*Dx*Dz+9*x^2*z^2*Dz^2+40*x^3*Dx+63*x^2*z*Dz+72\
   *x^2+12*Dx
↦    _[7]=3*y*z^4*Dz^2-2*x^4*Dx*Dy-6*x^3*z*Dy*Dz+21*y*z^3*Dz-12*x^3*Dy+24*y\
   *z^2-6*Dy
↦    _[8]=3*z^5*Dz^3-2*x^4*Dx*Dy^2-6*x^3*z*Dy^2*Dz+30*z^4*Dz^2-12*x^3*Dy^2+\
   66*z^3*Dz+24*z^2-6*Dy^2
↦ [2]:
↦    2
```

See also: Section 7.5.5.3 [DLoc], page 423; Section 7.5.5.5 [DLoc0], page 424; Section 7.5.5.4 [SDLoc], page 424.

7.5.14.2 WeylClosure

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: WeylClosure(I); I an ideal

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Moreover, assume that the holonomic rank of W/I is finite.

Return: ideal, the Weyl closure of I

Remarks: The Weyl closure of a left ideal I in the Weyl algebra W is defined to be the intersection of I regarded as left ideal in the rational Weyl algebra K(x(1..n))<D(1..n)> with the polynomial Weyl algebra W.
Reference: (Tsa), Algorithm 2.2.4

Note: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
// (OTW), Example 8
ring r = 0,(x,y,z,Dx,Dy,Dz),dp;
def D3 = Weyl();
setring D3;
poly f = x^3-y^2*z^2;
ideal I = f^2*Dx + 3*x^2, f^2*Dy-2*y*z^2, f^2*Dz-2*y^2*z;
// I annihilates exp(1/f)
WeylClosure(I);
↦ _[1]=y*Dy-z*Dz
↦ _[2]=2*y*z^2*Dx+3*x^2*Dy
↦ _[3]=2*y^2*z*Dx+3*x^2*Dz
↦ _[4]=2*z^3*Dx*Dz+3*x^2*Dy^2+2*z^2*Dx
↦ _[5]=4*x^4*Dx^2+12*x^3*z*Dx*Dz+9*x^2*z^2*Dz^2+16*x^3*Dx+27*x^2*z*Dz+12*Dx
↦ _[6]=3*y*z^4*Dz^2-2*x^4*Dx*Dy-6*x^3*z*Dy*Dz+9*y*z^3*Dz-6*Dy
↦ _[7]=3*y^2*z^3*Dz-2*x^4*Dx-6*x^3*z*Dz-6
↦ _[8]=3*z^5*Dz^3-2*x^4*Dx*Dy^2-6*x^3*z*Dy^2*Dz+18*z^4*Dz^2+18*z^3*Dz-6*Dy^\
   2
```

See also: Section 7.5.14.3 [WeylClosure1], page 524.

7.5.14.3 WeylClosure1

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: WeylClosure1(L); L a poly

Assume: The basering is the first Weyl algebra D=K<x,d|dx=xd+1> over a field K of characteristic 0.

Return: ideal, the Weyl closure of the principal left ideal generated by L

Remarks: The Weyl closure of a left ideal I in the Weyl algebra D is defined to be the intersection of I regarded as left ideal in the rational Weyl algebra K(x)<d> with the polynomial Weyl algebra D.
Reference: (Tsa), Algorithm 1.2.4

Note: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,Dx),dp;
def W = Weyl();
setring W;
poly L = (x^3+2)*Dx-3*x^2;
WeylClosure1(L);
↦ _[1]=x^3*Dx-3*x^2+2*Dx
↦ _[2]=x^2*Dx^2-2*x*Dx
↦ _[3]=x^2*Dx+Dx^2-3*x
↦ _[4]=x*Dx^2-2*Dx
L = (x^4-4*x^3+3*x^2)*Dx^2+(-6*x^3+20*x^2-12*x)*Dx+(12*x^2-32*x+12);
WeylClosure1(L);
↦ _[1]=x^4*Dx^2-4*x^3*Dx^2-6*x^3*Dx+3*x^2*Dx^2+20*x^2*Dx+12*x^2-12*x*Dx-32*\
   x+12
↦ _[2]=x^2*Dx^3-21/10*x^2*Dx^2+3/10*x*Dx^3-6/5*x*Dx^2+63/5*x*Dx-3/5*Dx^2-12\
   /5*Dx-126/5
↦ _[3]=x^3*Dx^2-43/10*x^2*Dx^2+9/10*x*Dx^3-6*x^2*Dx+12/5*x*Dx^2+109/5*x*Dx-\
   9/5*Dx^2+12*x-36/5*Dx-178/5
↦ _[4]=x^3*Dx^3-48/5*x^2*Dx^2+9/5*x*Dx^3+24/5*x*Dx^2+198/5*x*Dx-18/5*Dx^2-7\
   2/5*Dx-336/5
↦ _[5]=x^3*Dx^4-4*x^2*Dx^4+2*x^2*Dx^3+3*x*Dx^4-69/10*x^2*Dx^2+67/10*x*Dx^3-\
   24/5*x*Dx^2-3*Dx^3+207/5*x*Dx-27/5*Dx^2-18/5*Dx-414/5
↦ _[6]=x^3*Dx^6+8/3*x^3*Dx^5-4*x^2*Dx^6-2/3*x^2*Dx^5+3*x*Dx^6+16*x^2*Dx^4-2\
   0*x*Dx^5-92/3*x*Dx^4+12*Dx^5+126/5*x^2*Dx^2-258/5*x*Dx^3-168/5*x*Dx^2+92/\
   3*Dx^3-756/5*x*Dx+356/5*Dx^2+504/5*Dx+1512/5
```

See also: Section 7.5.14.2 [WeylClosure], page 524.

7.5.14.4 holonomicRank

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: holonomicRank(I[,e]); I ideal, e optional int

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Return: int, the holonomic rank of I

Remarks: The holonomic rank of I is defined to be the K(x(1..n))-dimension of the module W/WI, where W is the rational Weyl algebra K(x(1..n))<D(1..n)>.
If this dimension is infinite, -1 is returned.

Note: If e<>0, `std` is used for Groebner basis computations, otherwise (and by default) `slimgb` is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
// (OTW), Example 8
ring r3 = 0,(x,y,z,Dx,Dy,Dz),dp;
def D3 = Weyl();
setring D3;
poly f = x^3-y^2*z^2;
ideal I = f^2*Dx+3*x^2, f^2*Dy-2*y*z^2, f^2*Dz-2*y^2*z;
// I annihilates exp(1/f)
holonomicRank(I);
↦ 1
```

7.5.14.5 DsingularLocus

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: DsingularLocus(I); I ideal

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Return: ideal, describing the singular locus of the D-module D/I

Note: If printlevel>=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed

Example:

```
LIB "dmodloc.lib";
// (OTW), Example 8
ring @D3 = 0,(x,y,z,Dx,Dy,Dz),dp;
def D3 = Weyl();
setring D3;
poly f = x^3-y^2*z^2;
ideal I = f^2*Dx + 3*x^2, f^2*Dy-2*y*z^2, f^2*Dz-2*y^2*z;
// I annihilates exp(1/f)
DsingularLocus(I);
↦ _[1]=y^2*z^2-x^3
```

7.5.14.6 polSol

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: polSol(I[,w,m]); I ideal, w optional intvec, m optional int

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Moreover, assume that I is holonomic.

Return: ideal, a basis of the polynomial solutions to the given system of linear PDEs with polynomial coefficients, encoded via I

Remarks: If w is given, w should consist of n strictly negative entries. Otherwise and by default, w is set to -1:n.
In this case, w is used as weight vector for the computation of a b-function.

If m is given, m is assumed to be the minimal integer root of the b-function of I w.r.t. w. Note that this assumption is not checked.
Reference: (OTT), Algorithm 2.4

Note: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def W = Weyl();
setring W;
poly tx,ty = x*Dx, y*Dy;
ideal I =      // Appel F1 with parameters (2,-3,-2,5)
tx*(tx+ty+4)-x*(tx+ty+2)*(tx-3),
ty*(tx+ty+4)-y*(tx+ty+2)*(ty-2),
(x-y)*Dx*Dy+2*Dx-3*Dy;
intvec w = -1,-1;
polSol(I,w);
↦ _[1]=10*x^3*y^2-30*x^3*y-45*x^2*y^2+24*x^3+144*x^2*y+72*x*y^2-126*x^2-252\
   *x*y-42*y^2+252*x+168*y-210
```

See also: Section 7.5.14.7 [polSolFiniteRank], page 527; Section 7.5.14.8 [ratSol], page 528.

7.5.14.7 polSolFiniteRank

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: polSolFiniteRank(I[,w]); I ideal, w optional intvec

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Moreover, assume that I is of finite holonomic rank.

Return: ideal, a basis of the polynomial solutions to the given system of linear PDEs with polynomial coefficients, encoded via I

Remarks: If w is given, w should consist of n strictly negative entries. Otherwise and by default, w is set to -1:n.
In this case, w is used as weight vector for the computation of a b-function.
Reference: (OTT), Algorithm 2.6

Note: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def W = Weyl();
setring W;
poly tx,ty = x*Dx, y*Dy;
ideal I =      // Appel F1 with parameters (2,-3,-2,5)
tx*(tx+ty+4)-x*(tx+ty+2)*(tx-3),
ty*(tx+ty+4)-y*(tx+ty+2)*(ty-2),
(x-y)*Dx*Dy+2*Dx-3*Dy;
```

```
intvec w = -1,-1;
polSolFiniteRank(I,w);
↦ _[1]=10*x^3*y^2-30*x^3*y-45*x^2*y^2+24*x^3+144*x^2*y+72*x*y^2-126*x^2-252\
   *x*y-42*y^2+252*x+168*y-210
```

See also: Section 7.5.14.6 [polSol], page 526; Section 7.5.14.8 [ratSol], page 528.

7.5.14.8 ratSol

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: ratSol(I); I ideal

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Moreover, assume that I is holonomic.

Return: module, a basis of the rational solutions to the given system of linear PDEs with polynomial coefficients, encoded via I Note that each entry has two components, the first one standing for the enumerator, the second one for the denominator.

Remarks: Reference: (OTT), Algorithm 3.10

Note: If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def W = Weyl();
setring W;
poly tx,ty = x*Dx, y*Dy;
ideal I =      // Appel F1 with parameters (3,-1,1,1) is a solution
tx*(tx+ty)-x*(tx+ty+3)*(tx-1),
ty*(tx+ty)-y*(tx+ty+3)*(ty+1);
module M = ratSol(I);
// We obtain a basis of the rational solutions to I represented by a
// module / matrix with two rows.
// Each column of the matrix represents a rational function, where
// the first row correspond to the enumerator and the second row to
// the denominator.
print(M);
↦ x-y,              x,
↦ y^4-3*y^3+3*y^2-y,y
```

See also: Section 7.5.14.6 [polSol], page 526; Section 7.5.14.7 [polSolFiniteRank], page 527.

7.5.14.9 bfctBound

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: bfctBound (I,f[,primdec]); I ideal, f poly, primdec optional string

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by

x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i). Moreover, assume that I is holonomic.

Return: list of roots (of type ideal) and multiplicities (of type intvec) of a multiple of the b-function for f^s*u at a generic root of f. Here, u stands for [1] in D/I.

Remarks: Reference: (OTT), Algorithm 3.4

Note: This procedure requires to compute a primary decomposition in a commutative ring. The optional string primdec can be used to specify the algorithm to do so. It may either be 'GTZ' (Gianni, Trager, Zacharias) or 'SY' (Shimoyama, Yokoyama). By default, 'GTZ' is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def W = Weyl();
setring W;
poly tx,ty = x*Dx, y*Dy;
ideal I =      // Appel F1 with parameters (2,-3,-2,5)
tx*(tx+ty+4)-x*(tx+ty+2)*(tx-3),
ty*(tx+ty+4)-y*(tx+ty+2)*(ty-2),
(x-y)*Dx*Dy+2*Dx-3*Dy;
kill tx,ty;
poly f = x-1;
bfctBound(I,f);
↦ [1]:
↦    _[1]=-1
↦    _[2]=-7
↦ [2]:
↦    1,1
```

See also: Section D.6.13.4 [bernstein], page 1705; Section 7.5.2.1 [bfct], page 376; Section 7.5.2.3 [bfctAnn], page 378.

7.5.14.10 annRatSyz

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: annRatSyz(f,g[,db,eng]); f, g polynomials, db,eng optional integers

Assume: The basering is commutative and over a field of characteristic 0.

Return: ring (a Weyl algebra) containing an ideal 'LD', which is (part of) the annihilator of the rational function g/f in the corresponding Weyl algebra

Remarks: This procedure uses the computation of certain syzygies. One can obtain the full annihilator by computing the Weyl closure of the ideal LD.

Note: Activate the output ring with the `setring` command. In the output ring, the ideal 'LD' (in Groebner basis) is (part of) the annihilator of g/f.
If db>0 is given, operators of order up to db are considered, otherwise, and by default, a minimal holonomic solution is computed.
If eng<>0, `std` is used for Groebner basis computations, otherwise, and by default, `slimgb` is used.

If printlevel =1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "dmodloc.lib";
// printlevel = 3;
ring r = 0,(x,y),dp;
poly f = 2*x*y; poly g = x^2 - y^3;
def A = annRatSyz(f,g);   // compute a holonomic solution
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
LD;
↦ LD[1]=3*x*Dx+2*y*Dy+1
↦ LD[2]=y^4*Dy-x^2*y*Dy+2*y^3+x^2
setring r;
def B = annRatSyz(f,g,5); // compute a solution up to degree 5
setring B;
LD; // this is the full annihilator as we will check below
↦ LD[1]=3597*y^2*Dx^2*Dy-866*x*Dx*Dy^2-2176*y*Dy^3+10791*y*Dx^2-644*Dy^2
↦ LD[2]=111*y^3*Dx^2+2760*x^2*Dx^2+1470*x*y*Dx*Dy-296*y^2*Dy^2+4050*x*Dx
↦ LD[3]=3*x*Dx+2*y*Dy+1
↦ LD[4]=y^3*Dy^2-x^2*Dy^2+6*y^2*Dy+6*y
↦ LD[5]=y^4*Dy-x^2*y*Dy+2*y^3+x^2
setring r;
def C = annRat(f,g); setring C;
LD; // the full annihilator
↦ LD[1]=3*y^2*Dx^2*Dy+2*x*Dx*Dy^2+9*y*Dx^2+4*Dy^2
↦ LD[2]=3*y^3*Dx^2-10*x*y*Dx*Dy-8*y^2*Dy^2+10*x*Dx
↦ LD[3]=y^3*Dy^2-x^2*Dy^2-6*x*y*Dx+2*y^2*Dy+4*y
↦ LD[4]=3*x*Dx+2*y*Dy+1
↦ LD[5]=y^4*Dy-x^2*y*Dy+2*y^3+x^2
ideal BLD = imap(B,LD);
NF(LD,std(BLD));
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
```

See also: Section 7.5.5.1 [annPoly], page 421; Section 7.5.5.2 [annRat], page 422.

7.5.14.11 dmodGeneralAssumptionCheck

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: dmodGeneralAssumptionCheck();

Return: nothing, but checks general assumptions on the basering

Note: This procedure checks the following conditions on the basering R and prints an error message if any of them is violated:
- R is the n-th Weyl algebra over a field of characteristic 0,
- R is not a qring,
- for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by
x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,D),dp;
dmodGeneralAssumptionCheck(); // prints error message
↦    ? Basering is not a Weyl algebra
↦    ? leaving dmodloc.lib::dmodGeneralAssumptionCheck (0)
def W = Weyl();
setring W;
dmodGeneralAssumptionCheck(); // returns nothing
```

7.5.14.12 extendWeyl

Procedure from library dmodloc.lib (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: extendWeyl(S); S string or list of strings

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Return: ring, Weyl algebra extended by vars given by S

Example:

```
LIB "dmodloc.lib";
ring @D2 = 0,(x,y,Dx,Dy),dp;
def D2 = Weyl();
setring D2;
def D3 = extendWeyl("t");
setring D3; D3;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    t x y Dt Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dtt=t*Dt+1
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
list L = "u","v";
def D5 = extendWeyl(L);
setring D5;
D5;
↦ // coefficients: QQ
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    u v t x y Du Dv Dt Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Duu=u*Du+1
↦ //    Dvv=v*Dv+1
↦ //    Dtt=t*Dt+1
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
```

7.5.14.13 polyVars

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: polyVars(f,v); f poly, v intvec

Return: int, 1 if f contains only variables indexed by v, 0 otherwise

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,z),dp;
poly f = y^2+zy;
intvec v = 1,2;
polyVars(f,v); // does f depend only on x,y?
↦ 0
v = 2,3;
polyVars(f,v); // does f depend only on y,z?
↦ 1
```

7.5.14.14 monomialInIdeal

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: monomialInIdeal(I); I ideal

Return: ideal consisting of all monomials appearing in generators of ideal

Examlpe: example monomialInIdeal; shows examples

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
ideal I = x2+5x3y7, x-x2-6xy;
monomialInIdeal(I);
↦ _[1]=x3y7
↦ _[2]=x2
↦ _[3]=xy
↦ _[4]=x
```

7.5.14.15 vars2pars

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: vars2pars(v); v intvec

Assume: The basering is commutative.

Return: ring with variables specified by v converted into parameters

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,z,a,b,c),dp;
intvec v = 4,5,6;
def R = vars2pars(v);
setring R;
R;
↦ // coefficients: QQ(a, b, c)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
v = 1,2;
def RR = vars2pars(v);
setring RR;
RR;
↦ // coefficients: QQ(a, b, c, x, y)
↦ // number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    z
↦ //        block   2 : ordering C
```

7.5.14.16 minIntRoot2

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: minIntRoot2(L); L list

Assume: L is the output of bFactor.

Return: int, the minimal integer root in a list of roots

Example:

```
LIB "dmodloc.lib";
ring r = 0,x,dp;
poly f = x*(x+1)*(x-2)*(x-5/2)*(x+5/2);
list L = bFactor(f);
minIntRoot2(L);
↦ -1
```

See also: Section 7.5.5.24 [bFactor], page 441; Section 7.5.14.17 [maxIntRoot], page 533; Section 7.5.4.22 [minIntRoot], page 419.

7.5.14.17 maxIntRoot

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: maxIntRoot(L); L list

Assume: L is the output of bFactor.

Return: int, the maximal integer root in a list of roots

Example:

```
LIB "dmodloc.lib";
ring r = 0,x,dp;
poly f = x*(x+1)*(x-2)*(x-5/2)*(x+5/2);
list L = bFactor(f);
```

```
maxIntRoot(L);
↦ 2
```

See also: Section 7.5.5.24 [bFactor], page 441; Section 7.5.14.16 [minIntRoot2], page 533.

7.5.14.18 dmodAction

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: dmodAction(id,f[,v]); id ideal or poly, f poly, v optional intvec

Assume: If v is not given, the basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Otherwise, v is assumed to specify positions of variables, which form a Weyl algebra as a subalgebra of the basering:
If size(v) equals 2*n, then bracket(var(v[i]),var(v[j])) must equal 1 if and only if j equals i+n, and 0 otherwise, for all 1<=i,j<=n.
Further, assume that f does not contain any D(i).

Return: same type as id, the result of the natural D-module action of id on f

Note: The assumptions made are not checked.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,z),dp;
poly f = x^2*z - y^3;
def A = annPoly(f);
setring A;
poly f = imap(r,f);
dmodAction(LD,f);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
↦ _[6]=0
↦ _[7]=0
↦ _[8]=0
↦ _[9]=0
↦ _[10]=0
↦ _[11]=0
↦ _[12]=0
↦ _[13]=0
poly P = y*Dy+3*z*Dz-3;
dmodAction(P,f);
↦ 0
dmodAction(P[1],f);
↦ -3*y^3
```

7.5.14.19 dmodActionRat

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: dmodActionRat(id,w); id ideal or poly, f vector

Assume: The basering is the n-th Weyl algebra W over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).
Further, assume that w has exactly two components, second one not 0, and that w does not contain any D(i).

Return: same type as id, the result of the natural D-module action of id on the rational function w[1]/w[2]

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
poly f = 2*x;  poly g = y;
def A = annRat(f,g); setring A;
poly f = imap(r,f); poly g = imap(r,g);
vector v = [f,g]; // represents f/g
// x and y act by multiplication
dmodActionRat(x,v);
↦ _[1]=2*x^2*gen(1)+y*gen(2)
dmodActionRat(y,v);
↦ _[1]=2*x*gen(1)+gen(2)
// Dx and Dy act by partial derivation
dmodActionRat(Dx,v);
↦ _[1]=y*gen(2)+2*gen(1)
dmodActionRat(Dy,v);
↦ _[1]=y^2*gen(2)-2*x*gen(1)
dmodActionRat(x*Dx+y*Dy,v);
↦ _[1]=gen(2)
setring r;
f = 2*x*y; g = x^2 - y^3;
def B = annRat(f,g); setring B;
poly f = imap(r,f); poly g = imap(r,g);
vector v = [f,g];
dmodActionRat(LD,v); // hence LD is indeed the annihilator of f/g
↦ _[1]=gen(2)
↦ _[2]=gen(2)
↦ _[3]=gen(2)
↦ _[4]=gen(2)
↦ _[5]=gen(2)
```

7.5.14.20 simplifyRat

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: simplifyRat(v); v vector

Assume: Assume that v has exactly two components, second one not 0.

Return: vector, representing simplified rational function v[1]/v[2]

Note: Possibly present non-commutative relations of the basering are ignored.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
vector v = [x2-1,x+1];
simplifyRat(v);
↦ x*gen(1)+gen(2)-gen(1)
simplifyRat(v) - [x-1,1];
↦ 0
```

7.5.14.21 addRat

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: addRat(v,w); v,w vectors

Assume: Assume that v,w have exactly two components, second ones not 0.

Return: vector, representing rational function (v[1]/v[2])+(w[1]/w[2])

Note: Possibly present non-commutative relations of the basering are ignored.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
vector v = [x,y];
vector w = [y,x];
addRat(v,w);
↦ x2*gen(1)+xy*gen(2)+y2*gen(1)
addRat(v,w) - [x2+y2,xy];
↦ 0
```

7.5.14.22 multRat

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: multRat(v,w); v,w vectors

Assume: Assume that v,w have exactly two components, second ones not 0.

Return: vector, representing rational function (v[1]/v[2])*(w[1]/w[2])

Note: Possibly present non-commutative relations of the basering are ignored.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
vector v = [x,y];
vector w = [y,x];
multRat(v,w);
↦ gen(2)+gen(1)
multRat(v,w) - [1,1];
↦ 0
```

7.5.14.23 diffRat

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: diffRat(v,j); v vector, j int

Assume: Assume that v has exactly two components, second one not 0.

Return: vector, representing rational function derivative of rational function (v[1]/v[2]) w.r.t. var(j)

Note: Possibly present non-commutative relations of the basering are ignored.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y),dp;
vector v = [x,y];
diffRat(v,1);
↦ y*gen(2)+gen(1)
diffRat(v,1) - [1,y];
↦ 0
diffRat(v,2);
↦ y2*gen(2)-x*gen(1)
diffRat(v,2) - [-x,y2];
↦ 0
```

7.5.14.24 commRing

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: commRing();

Return: ring, basering without non-commutative relations

Example:

```
LIB "dmodloc.lib";
def W = makeWeyl(3);
setring W; W;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) D(1) D(2) D(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    D(1)x(1)=x(1)*D(1)+1
↦ //    D(2)x(2)=x(2)*D(2)+1
↦ //    D(3)x(3)=x(3)*D(3)+1
def W2 = commRing();
setring W2; W2;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) D(1) D(2) D(3)
↦ //        block   2 : ordering C
ring r = 0,(x,y),dp;
def r2 = commRing(); // same as r
setring r2; r2;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
```

7.5.14.25 rightNFWeyl

Procedure from library `dmodloc.lib` (see Section 7.5.14 [dmodloc_lib], page 522).

Usage: rightNFWeyl(id,k); id ideal or poly, k int

Assume: The basering is the n-th Weyl algebra over a field of characteristic 0 and for all 1<=i<=n the identity
var(i+n)*var(i)=var(i)*var(i+1)+1 holds, i.e. the sequence of variables is given by x(1),...,x(n),D(1),...,D(n), where D(i) is the differential operator belonging to x(i).

Return: same type as id, the right normal form of id with respect to the principal right ideal generated by the k-th variable

Note: No Groebner basis computation is used.

Example:

```
LIB "dmodloc.lib";
ring r = 0,(x,y,Dx,Dy),dp;
def W = Weyl();
setring W;
ideal I = x^3*Dx^3, y^2*Dy^2, x*Dy, y*Dx;
rightNFWeyl(I,1); // right NF wrt principal right ideal x*W
↦ _[1]=0
↦ _[2]=y^2*Dy^2
↦ _[3]=0
↦ _[4]=y*Dx
rightNFWeyl(I,3); // right NF wrt principal right ideal Dx*W
↦ _[1]=-6
↦ _[2]=y^2*Dy^2
↦ _[3]=x*Dy
↦ _[4]=0
rightNFWeyl(I,2); // right NF wrt principal right ideal y*W
↦ _[1]=x^3*Dx^3
↦ _[2]=0
↦ _[3]=x*Dy
↦ _[4]=0
rightNFWeyl(I,4); // right NF wrt principal right ideal Dy*W
↦ _[1]=x^3*Dx^3
↦ _[2]=2
↦ _[3]=0
↦ _[4]=y*Dx
poly p = x*Dx+1;
rightNFWeyl(p,1); // right NF wrt principal right ideal x*W
↦ 1
```

7.5.15 ncfrac_lib

Library: ncfrac.lib

Purpose: object-oriented interface for olga.lib

Author: Johannes Hoffmann, email: johannes.hoffmann at math.rwth-aachen.de

Overview: This library introduces a new type: ncfrac.
This type wraps the data defining a (non-commutative) fraction in an Ore localization

of a G-algebra as in olga.lib.
An element of type ncfrac has five members:
- polys lnum, lden, rnum, rden
- ncloc loc

Operations:

string(ncfrac);
give a string representation of the data describing the fraction print(ncfrac);
prints the string representation of the fraction
status(ncfrac);
report on the status/validity of the fraction
test(ncfrac);
check if the fraction is valid

Infix operations:

ncfrac == ncfrac;
compare two fractions
ncfrac != ncfrac;
compare two fractions
ncfrac + ncfrac;
add two fractions
ncfrac - ncfrac
subtract two fractions
ncfrac * ncfrac
multiply two fractions
ncfrac / ncfrac
divide two fractions
ncfrac = int/number/poly
create a fraction with:
- left and right denominator equal to 1
- left and right numerator determined by the input - localization data describing the trivial monoidal localization at 1 ncfrac = vector
create a fraction from a vector v with unspecified localization such that lden,lnum,rnum,rden = v[1],v[2],v[3],v[4]
(note: without specifying a localization afterwards this results is an invalid fraction)
ncfrac = list
create a fraction from a list L as follows: - try to create a fraction from L[1] as above
- if L[2] is of type ncloc set the localization of the fraction to L[2]

Procedures:

7.5.15.1 hasLeftDenom

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: hasLeftDenom(frac), ncfrac frac

Purpose: checks if frac has a left representation

Return: int, 1 if frac has a left representation, 0 otherwise

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/hasLeftDenom.s\
   ing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = ideal(x-3,y+7);
ncfrac noLeft = list([0,0,3*y*Dx,x+2], loc);
hasLeftDenom(noLeft);
↦ 0
ncfrac left = list([1,Dx,Dx,1], loc);
hasLeftDenom(left);
↦ 1
```

7.5.15.2 hasRightDenom

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: hasRightDenom(frac), ncfrac frac

Purpose: checks if frac has a right representation

Return: int, 1 if frac has a right representation, 0 otherwise

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/hasRightDenom.\
   sing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = ideal(x-3,y+7);
ncfrac noRight = list([x+2,3*y*Dx,0,0], loc);
hasRightDenom(noRight);
↦ 0
ncfrac right = list([1,Dx,Dx,1], loc);
hasRightDenom(right);
↦ 1
```

7.5.15.3 isZeroNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: isZeroNcfrac(frac), ncfrac frac

Purpose: checks if frac is zero

Return: int, 1 if frac is zero, 0 otherwise

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/isZeroNcfrac.s\
   ing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
```

```
ring Q = (0,q),(x,y,Qx,Qy),dp;
matrix C[4][4] = UpOneMatrix(4);
C[1,3] = q;
C[2,4] = q;
def ncQ = nc_algebra(C,0);
setring ncQ;
ncloc loc = intvec(2);
ncfrac frac = list([y^2+7*y+1,0,0,0], loc);
isZeroNcfrac(frac);
↦ 1
frac.lnum = 42*y*Qy+7*Qx+3*x+7;
isZeroNcfrac(frac);
↦ 0
```

7.5.15.4 isOneNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: isOneNcfrac(frac), ncfrac frac

Purpose: checks if frac is one

Return: int, 1 if frac is one, 0 otherwise

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/isOneNcfrac.si\
   ng:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring Q = (0,q),(x,y,Qx,Qy),dp;
matrix C[4][4] = UpOneMatrix(4);
C[1,3] = q;
C[2,4] = q;
def ncQ = nc_algebra(C,0);
setring ncQ;
ncloc loc = intvec(2);
ncfrac frac = list([y^2+7*y+1,y^2+7*y+1,0,0], loc);
isOneNcfrac(frac);
↦ 1
frac.lnum = 42*y*Qy+7*Qx+3*x+7;
isOneNcfrac(frac);
↦ 0
```

7.5.15.5 zeroNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: zeroNcfrac(loc), ncloc loc

Purpose: returns the zero fraction in the localization loc

Return: ncfrac

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/zeroNcfrac.sin\
   g:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = ideal(x-53,y-7);
zeroNcfrac(loc);
↦ left repr.: (1,0)
↦ right repr.: (0,1)
↦
```

7.5.15.6 oneNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: oneNcfrac(loc), ncloc loc

Purpose: returns the one fraction in the localization loc

Return: ncfrac

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/oneNcfrac.sing\
   :1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = ideal(x-42,y-17);
oneNcfrac(loc);
↦ left repr.: (1,1)
↦ right repr.: (1,1)
↦
```

7.5.15.7 ensureLeftNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: ensureLeftNcfrac(frac), ncfrac frac

Purpose: ensures that frac has a left representation (by computing it if not alreaDy known)

Return: ncfrac, a representation of frac which has a left representation

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/ensureLeftNcfr\
   ac.sing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
```

```
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x+3;
poly g2 = x*y;
list L = g1,g2;
ncloc loc0 = L;
poly g = g1^2*g2;
poly f = Dx;
ncfrac frac0 = [0,0,f,g];
frac0.loc = loc0;
ncfrac rm = ensureLeftNcfrac(frac0);
print(rm);
↦ left repr.: (x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4,x^5*y^3\
   *Dx+6*x^4*y^3*Dx-3*x^4*y^3+9*x^3*y^3*Dx-12*x^3*y^3-9*x^2*y^3)
↦ right repr.: (Dx,x^3*y+6*x^2*y+9*x*y)
rm.lnum*g-rm.lden*f;
↦ 0
// geometric localization
ncloc loc1 = ideal(x-1,y-3);
f = Dx;
g = x^2+y;
ncfrac frac1 = [0,0,f,g];
frac1.loc = loc1;
ncfrac rg = ensureLeftNcfrac(frac1);
print(rg);
↦ left repr.: (x^4+2*x^2*y+y^2,x^2*Dx+y*Dx-2*x)
↦ right repr.: (Dx,x^2+y)
rg.lnum*g-rg.lden*f;
↦ 0
// rational localization
intvec rat = 1;
ncloc loc2 = rat;
f = Dx+Dy;
g = x;
ncfrac frac2 = [0,0,f,g];
frac2.loc = loc2;
ncfrac rr = ensureLeftNcfrac(frac2);
print(rr);
↦ left repr.: (x^2,x*Dx+x*Dy-1)
↦ right repr.: (Dx+Dy,x)
rr.lnum*g-rr.lden*f;
↦ 0
```

7.5.15.8 ensureRightNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: ensureLeftNcfrac(frac), ncfrac frac

Purpose: ensures that frac has a right representation (by computing it if not alreaDy known)

Return: ncfrac, a representation of frac which has a right representation

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/ensureRightNcf\
   rac.sing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g = x;
poly f = Dx;
ncloc loc0 = g;
ncfrac frac0 = [g,f,0,0];
frac0.loc = loc0;
ncfrac rm = ensureRightNcfrac(frac0);
print(rm);
↦ left repr.: (x,Dx)
↦ right repr.: (x*Dx+2,x^2)
f*rm.rden-g*rm.rnum;
↦ 0
// geometric localization
g = x+y;
f = Dx+Dy;
ncloc loc1 = ideal(x-1,y-3);
ncfrac frac1 = [g,f,0,0];
frac1.loc = loc1;
ncfrac rg = ensureRightNcfrac(frac1);
print(rg);
↦ left repr.: (x+y,Dx+Dy)
↦ right repr.: (x*Dx+y*Dx+x*Dy+y*Dy+4,x^2+2*x*y+y^2)
f*rg.rden-g*rg.rnum;
↦ 0
// rational localization
intvec rat = 1;
f = Dx+Dy;
g = x;
```

```
ncloc loc2 = rat;
ncfrac frac2 = [g,f,0,0];
frac2.loc = loc2;
ncfrac rr = ensureRightNcfrac(frac2);
print(rr);
↦ left repr.: (x,Dx+Dy)
↦ right repr.: (x*Dx+x*Dy+2,x^2)
f*rr.rden-g*rr.rnum;
↦ 0
```

7.5.15.9 negateNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: negateNcfrac(frac), ncfrac frac

Purpose: compute the negative (i.e. additive inverse) of frac

Return: ncfrac

Note: returns (-1)*frac

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/negateNcfrac.s\
   ing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
poly g = x*y^2+4*x+7*y-98;
ncloc loc = g;
ncfrac frac = list([g, 13*x^2], loc);
frac;
↦ left repr.: (x*y^2+4*x+7*y-98,13*x^2)
↦ right repr.: (0,0)
↦
ncfrac negFrac = negateNcfrac(frac);
negFrac;
↦ left repr.: (x*y^2+4*x+7*y-98,-13*x^2)
↦ right repr.: (0,0)
↦
frac + negFrac;
↦ left repr.: (x*y^2+4*x+7*y-98,0)
↦ right repr.: (0,0)
↦
```

7.5.15.10 isInvertibleNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: isInvertibleNcfrac(frac), ncfrac frac

Purpose: checks if frac is invertible

Return: int, 1 if frac is invertible, 0 otherwise

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/isInvertibleNc\
   frac.sing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = intvec(2);
ncfrac frac = list([y,y+1,0,0], loc);
isInvertibleNcfrac(frac);
↦ 1
frac = list([y,x+1,0,0], loc);
isInvertibleNcfrac(frac);
↦ 0
```

7.5.15.11 invertNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: invertNcfrac(frac), ncfrac frac

Purpose: compute the inverse of frac

Return: ncfrac

Note: returns the zero fraction if frac is not invertible

Example:

```
LIB "ncfrac.lib";
↦ // ** redefining testNcfrac (LIB "ncfrac.lib";) ./examples/invertNcfrac.s\
   ing:1
↦ // ** redefining testNcloc (    LIB "ncloc.lib";) ncfrac.lib::mod_init:11\
   3
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S;
ncloc loc = intvec(2);
ncfrac frac1 = list([y,y+1,0,0], loc);
// frac1 is invertible
ncfrac inv = invertNcfrac(frac1);
inv;
↦ left repr.: (y+1,y)
↦ right repr.: (0,0)
↦
ncfrac frac2 = list([y,x+1,0,0], loc);
// frac2 is not invertible
inv = invertNcfrac(frac2);
inv;
↦ left repr.: (1,0)
↦ right repr.: (0,1)
↦
```

7.5.15.12 testNcfrac

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: testNcfrac()

Purpose: execute a series of internal testing procedures

Return: nothing

Note:

7.5.15.13 testNcfracExamples

Procedure from library `ncfrac.lib` (see Section 7.5.15 [ncfrac_lib], page 538).

Usage: testNcfracExamples()

Purpose: execute the examples of all procedures in this library

Return: nothing

Note:

7.5.16 nchomolog_lib

Status: experimental

Library: nchomolog.lib

Purpose: Procedures for Noncommutative Homological Algebra

Authors: Viktor Levandovskyy levandov@math.rwth-aachen.de,
Christian Schilli, christian.schilli@rwth-aachen.de,
Gerhard Pfister, pfister@mathematik.uni-kl.de

Overview: In this library we present tools of homological algebra for finitely presented modules over GR-algebras.

Procedures:

7.5.16.1 ncExt_R

Procedure from library `nchomolog.lib` (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: ncExt_R(i, M); i int, M module

Compute: a presentation of Ext^i(M',R); for M'=coker(M).

Return: right module Ext, a presentation of Ext^i(M',R)

Example:

```
LIB "nchomolog.lib";
ring R     = 0,(x,y),dp;
poly F     = x2-y2;
def A = annfs(F);  setring A; // A is the 2nd Weyl algebra
matrix M[1][size(LD)] = LD; // ideal
print(M);
↦ y*Dx+x*Dy,x*Dx+y*Dy+2,x^2*Dy-y^2*Dy-2*y
print(ncExt_R(1,M)); // hence the Ext^1 is zero
↦ 1,0,
↦ 0,1
module E  = ncExt_R(2,M); // define the right module E
print(E); // E is in the opposite algebra
↦ 1, -x,    -y,
↦ Dx,y*Dy+1,x*Dy
def Aop = opposite(A);  setring Aop;
module Eop = oppose(A,E);
module T1  = ncExt_R(2,Eop);
setring A;
module T1 = oppose(Aop,T1);
print(T1); // this is a left module Ext^2(Ext^2(M,A),A)
↦ y*Dx+x*Dy,x*Dx+y*Dy+2,x^2*Dy-y^2*Dy-2*y
print(M); // it is known that M holonomic implies Ext^2(Ext^2(M,A),A) iso to M
↦ y*Dx+x*Dy,x*Dx+y*Dy+2,x^2*Dy-y^2*Dy-2*y
```

7.5.16.2 ncHom

Procedure from library nchomolog.lib (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: ncHom(M,N); M,N modules

Compute: A presentation of Hom(M',N'), M'=coker(M), N'=coker(N)

Assume: M' is a left module, N' is a centralizing bimodule

Note: ncHom(M,N) is a right module, hence a right presentation matrix is returned

Example:

```
LIB "nchomolog.lib";
ring A=0,(x,y,z),dp;
matrix M[3][3]=1,2,3,
4,5,6,
7,8,9;
matrix N[2][2]=x,y,
z,0;
module H = ncHom(M,N);
print(H);
↦ 0,0,0,0,y,x,
↦ 0,0,0,0,0,z,
↦ 1,0,0,0,0,0,
↦ 0,1,0,0,0,0,
↦ 0,0,1,0,0,0,
↦ 0,0,0,1,0,0
```

7.5.16.3 coHom

Procedure from library nchomolog.lib (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: coHom(A,k); A matrix, k int

Purpose: compute the matrix of a homomorphism Hom(R^k,A), where R is the basering. Let A be a matrix defining a map F1–>F2 of free R-modules, then the matrix of Hom(R^k,F1)–>Hom(R^k,F2) is computed.

Note: Both A and Hom(A,R^k) are matrices for either left or right R-module homomorphisms

Example:

```
LIB "nchomolog.lib";
ring A=0,(x,y,z),dp;
matrix M[3][3]=1,2,3,
4,5,6,
7,8,9;
module cM = coHom(M,2);
print(cM);
↦ 1,0,2,0,3,0,
↦ 0,1,0,2,0,3,
↦ 4,0,5,0,6,0,
↦ 0,4,0,5,0,6,
↦ 7,0,8,0,9,0,
↦ 0,7,0,8,0,9
```

7.5.16.4 contraHom

Procedure from library `nchomolog.lib` (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: contraHom(A,k); A matrix, k int

Return: matrix

Purpose: compute the matrix of a homomorphism Hom(A,R^k), where R is the basering. Let A be a matrix defining a map F1–>F2 of free R-modules, then the matrix of Hom(F2,R^k)–>Hom(F1,R^k) is computed.

Note: if A is matrix of a left (resp. right) R-module homomorphism, then Hom(A,R^k) is a right (resp. left) R-module R-module homomorphism

Example:

```
LIB "nchomolog.lib";
ring A=0,(x,y,z),dp;
matrix M[3][3]=1,2,3,
4,5,6,
7,8,9;
module cM = contraHom(M,2);
print(cM);
↦ 1,4,7,0,0,0,
↦ 2,5,8,0,0,0,
↦ 3,6,9,0,0,0,
↦ 0,0,0,1,4,7,
↦ 0,0,0,2,5,8,
↦ 0,0,0,3,6,9
```

7.5.16.5 dmodoublext

Procedure from library `nchomolog.lib` (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: dmodoublext(M [,i]); M module, i optional int

Compute: a presentation of Ext^i(Ext^i(M,D),D) for basering D

Return: left module

Note: by default, i is set to the integer part of the half of number of variables of D
for holonomic modules over Weyl algebra, the double ext is known to be holonomic left module

Example:

```
LIB "nchomolog.lib";
ring R   = 0,(x,y),dp;
poly F   = x3-y2;
def A    = annfs(F);
setring A;
dmodoublext(LD);
↦ _[1]=2*x*Dx+3*y*Dy+6
↦ _[2]=3*x^2*Dy+2*y*Dx
↦ _[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ _[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
LD;
↦ LD[1]=2*x*Dx+3*y*Dy+6
↦ LD[2]=3*x^2*Dy+2*y*Dx
↦ LD[3]=9*x*y*Dy^2-4*y*Dx^2+15*x*Dy
↦ LD[4]=27*y^2*Dy^3+8*y*Dx^3+135*y*Dy^2+105*Dy
// fancier example:
setring A;
ideal I = Dx*(x2-y3),Dy*(x2-y3);
I = groebner(I);
print(dmodoublext(I,1));
↦ y^3-x^2
print(dmodoublext(I,2));
↦ Dy,
↦ Dx
```

7.5.16.6 is_cenBimodule

Procedure from library `nchomolog.lib` (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: is_cenBimodule(M); M module

Compute: 1, if a module, presented by M can be centralizing in the sense of Artin and 0 otherwise

Note: only one condition for centralizing factor module can be checked algorithmically

Example:

```
LIB "nchomolog.lib";
def A = makeUsl2(); setring A;
poly p = 4*e*f + h^2-2*h; // generator of the center
matrix M[2][2] = p, p^2-7,0,p*(p+1);
is_cenBimodule(M); // M is centralizing
↦ 1
matrix N[2][2] = p, e*f,h,p*(p+1);
is_cenBimodule(N); // N is not centralizing
↦ 0
```

7.5.16.7 is_cenSubbimodule

Procedure from library `nchomolog.lib` (see Section 7.5.16 [nchomolog_lib], page 547).

Usage: is_cenSubbimodule(M); M module

Compute: 1, if a subbimodule, generated by the columns of M is centralizing in the sense of Artin and 0 otherwise

Example:

```
LIB "nchomolog.lib";
def A = makeUsl2(); setring A;
poly p = 4*e*f + h^2-2*h; // generator of the center
matrix M[2][2] = p, p^2-7,0,p*(p+1);
is_cenSubbimodule(M); // M is centralizing subbimodule
↦ 1
matrix N[2][2] = p, e*f,h,p*(p+1);
is_cenSubbimodule(N); // N is not centralizing subbimodule
↦ 0
```

7.5.17 ncloc_lib

Library: ncloc.lib

Purpose: Ore-localization in G-Algebras

Author: Johannes Hoffmann, email: johannes.hoffmann at math.rwth-aachen.de

Overview: This library introduces a new type: ncloc.
This type wraps the localization data defined as in olga.lib. An element of type ncloc has two members:
- int locType
- def locData

Operations:

string(ncloc);
give a string representation of the data describing the localization print(ncloc);
prints the string representation of the localization status(ncloc);
report on the status/validity of the localization test(ncloc);
check if the localization is valid

Infix operations:

ncloc == ncloc;
compare two localizations
ncloc != ncloc;
compare two localizations
ncloc = list/poly
create a monoidal localization from the given data ncloc = ideal
create a geometric localization from the given data ncloc = intvec
create a rational localization from the given data

Procedures:

7.5.17.1 isDenom

Procedure from library `ncloc.lib` (see Section 7.5.17 [ncloc_lib], page 551).

Usage: isDenom(p, loc), poly a, ncloc loc

Purpose: check if p is a valid denominator in the localization loc

Return: int

Note: returns 1 or 0, depending whether p is a valid denominator

Example:

```
LIB "ncloc.lib";
↦ // ** redefining testNcloc (LIB "ncloc.lib";) ./examples/isDenom.sing:1
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
ncloc loc;
poly g1 = x^2*y+x+2;
poly g2 = y^3+x*y;
list L = g1,g2;
loc = L;
poly g = g1^2*g2;
poly f = g - 1;
isDenom(g, loc);
↦ 1
isDenom(f, loc);
↦ 0
// geometrical localization
loc = ideal(x-1,y-3);
g = x^2+y-3;
f = (x-1)*g;
isDenom(g, loc);
↦ 1
isDenom(f, loc);
↦ 0
// rational localization
intvec v = 2;
loc = v;
g = y^5+17*y^2-4;
f = x*y;
isDenom(g, loc);
↦ 1
isDenom(f, loc);
↦ 0
```

7.5.17.2 testNcloc

Procedure from library `ncloc.lib` (see Section 7.5.17 [ncloc_lib], page 551).

Usage: testNcloc()

Purpose: execute a series of internal testing procedures

Return: nothing

Note:

7.5.17.3 testNclocExamples

Procedure from library `ncloc.lib` (see Section 7.5.17 [ncloc_lib], page 551).

Usage: testNclocExamples()

Purpose: execute the examples of all procedures in this library

Return: nothing

Note:

7.5.18 ncModslimgb_lib

Library: ncModslimgb.lib

Purpose: A library for computing Groebner bases over G-algebras defined over the rationals using modular techniques.

Authors: Wolfram Decker, Christian Eder, Viktor Levandovskyy, and Sharwan K. Tiwari shrawant@gmail.com

References:

Wolfram Decker, Christian Eder, Viktor Levandovskyy, and Sharwan K. Tiwari, Modular Techniques For Noncommutative Groebner Bases, https://link.springer.com/article/10.1007/s11786-019-00412-9 and https://arxiv.org/abs/1704.02852.

E. A. Arnold, Modular algorithms for computing Groebner bases. Journal of Symbolic Computation 35, 403-419 (2003).

N. Idrees, G. Pfister, S. Steidel, Parallelization of Modular Algorithms, Journal of Symbolic Computation 46, 672-684 (2011).

Procedures:

7.5.18.1 ncmodslimgb

Procedure from library `ncModslimgb.lib` (see Section 7.5.18 [ncModslimgb_lib], page 553).

Usage: ncmodslimgb(I[, exactness, ncores]); I ideal, optional integers exactness and n(umber of)cores

Return: ideal

Purpose: compute a left Groebner basis of I by modular approach

Assume: basering is a G-algebra; base field is prime field Q of rationals.

Note: - If the given algebra and ideal are graded (it is not checked by this command), then the computed Groebner basis will be exact. Otherwise, the result will be correct with a very high probability. - The optional parameter ‘exactness‘ justifies, whether the final (expensive) verification step will be performed or not (exactness=0, default value is 1). - The optional parameter ‘ncores‘ (default value is 1) provides an integer to use the number of cores (this must not exceed the number of available cores in the computing machine).

Example:

```
LIB "ncModslimgb.lib";
ring r = 0,(x,y),dp;
poly P = y^4+x^3+x*y^3; // a (3,4)-Reiffen curve
def A = Sannfs(P); setring A; // computed D-module data from P
ideal bs = LD, imap(r,P); // preparing the computation of the Bernstein-Sato polynomi
ideal I1 = ncmodslimgb(bs,0,2); // no final verification, use 2 cores
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
↦ // ** going to redefine the algebra structure
I1[1]; // the Bernstein-Sato polynomial of P, univariate in s
↦ s^7+7*s^6+499/24*s^5+815/24*s^4+227563/6912*s^3+43627/2304*s^2+4461779/74\
   6496*s+595595/746496
ideal I2 = ncmodslimgb(bs); // do the final verification, use 1 core (default)
I2[1]; // the Bernstein-Sato polynomial of P, univariate in s
↦ s^7+7*s^6+499/24*s^5+815/24*s^4+227563/6912*s^3+43627/2304*s^2+4461779/74\
   6496*s+595595/746496
```

7.5.19 ncpreim_lib

Library: ncpreim.lib

Purpose: Non-commutative elimination and preimage computations

Author: Daniel Andres, daniel.andres@math.rwth-aachen.de
Support: DFG Graduiertenkolleg 1632 ‘Experimentelle und konstruktive Algebra’

Overview: In G-algebras, elimination of variables is more involved than in the commutative case. One, not every subset of variables generates an algebra, which is again a G-algebra.

Two, even if the subset of variables in question generates an admissible subalgebra, there might be no admissible elimination ordering, i.e. an elimination ordering which also satisfies the ordering condition for G-algebras.

The difference between the procedure `eliminateNC` provided in this library and the procedure `eliminate (plural)` from the kernel is that eliminateNC will always find an admissible elimination if such one exists. Moreover, the use of `slimgb` for performing Groebner basis computations is possible.

As an application of the theory of elimination, the procedure `preimageNC` is provided, which computes the preimage of an ideal under a homomorphism f: A `->` B between G-algebras A and B. In contrast to the kernel procedure `preimage (plural)`, the assumption that A is commutative is not required.

References:

(BGL) J.L. Bueso, J. Gomez-Torrecillas, F.J. Lobillo: 'Re-filtering and exactness of the Gelfand-Kirillov dimension', Bull. Sci. math. 125, 8, 689-715, 2001.
(GML) J.I. Garcia Garcia, J. Garcia Miranda, F.J. Lobillo: 'Elimination orderings and localization in PBW algebras', Linear Algebra and its Applications 430(8-9), 2133-2148, 2009.
(Lev) V. Levandovskyy: 'Intersection of ideals with non-commutative subalgebras', ISSAC'06, 212-219, ACM, 2006.

Procedures: See also: Section D.4.7 [elim_lib], page 1058; Section 7.3.21 [preimage (plural)], page 354.

7.5.19.1 eliminateNC

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: eliminateNC(I,v,eng); I ideal, v intvec, eng optional int

Return: ideal, I intersected with the subring defined by the variables not index by the entries of v

Assume: The entries of v are in the range 1..nvars(basering) and the corresponding variables generate an admissible subalgebra.

Remarks: In order to determine the required elimination ordering, a linear programming problem is solved with the simplex algorithm.
Reference: (GML)
Unlike eliminate, this procedure will always find an elimination ordering, if such exists.

Note: If eng<>0, `std` is used for Groebner basis computations, otherwise (and by default) `slimgb` is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Example:

```
LIB "ncpreim.lib";
// (Lev): Example 2
ring r = 0,(a,b,x,d),Dp;
matrix D[4][4];
D[1,2] = 3*a; D[1,4] = 3*x^2;
D[2,3] = -x;  D[2,4] = d;     D[3,4] = 1;
def A = nc_algebra(1,D);
setring A; A;
```

```
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering Dp
↦ //                  : names    a b x d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=ab+3a
↦ //    da=ad+3x2
↦ //    xb=bx-x
↦ //    db=bd+d
↦ //    dx=xd+1
ideal I = a,x;
// Since d*a-a*d = 3*x^2, any admissible ordering has to satisfy
// x^2 < a*d, while any elimination ordering for {x,d} additionally
// has to fulfil a << x and a << d.
// Hence, the weight (0,0,1,1) is not an elimination weight for
// (x,d) and the call eliminate(I,x*d); will produce an error.
eliminateNC(I,3..4);
↦ _[1]=a
// This call uses the elimination weight (0,0,1,2), which works.
```

See also: Section 7.3.5 [eliminate (plural)], page 338.

7.5.19.2 preimageNC

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: preimageNC(A,f,J[,P,eng]); A ring, f map or ideal, J ideal, P optional string, eng optional int

Assume: f defines a map from A to the basering.

Return: nothing, instead exports an object 'preim' of type ideal to ring A, being the preimage of J under f.

Note: If P is given and not equal to the empty string, the preimage is exported to A under the name specified by P.
Otherwise (and by default), P is set to 'preim'.
If eng<>0, `std` is used for Groebner basis computations, otherwise (and by default) `slimgb` is used.
If printlevel=1, progress debug messages will be printed, if printlevel>=2, all the debug messages will be printed.

Remark: Reference: (Lev)

Example:

```
LIB "ncpreim.lib";
def A = makeUgl(3); setring A; A; // universal enveloping algebra of gl_3
↦ // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    e_1_1 e_1_2 e_1_3 e_2_1 e_2_2 e_2_3 e_3_1 \
   e_3_2 e_3_3
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    e_1_2e_1_1=e_1_1*e_1_2-e_1_2
↦ //    e_1_3e_1_1=e_1_1*e_1_3-e_1_3
↦ //    e_2_1e_1_1=e_1_1*e_2_1+e_2_1
↦ //    e_3_1e_1_1=e_1_1*e_3_1+e_3_1
↦ //    e_2_1e_1_2=e_1_2*e_2_1-e_1_1+e_2_2
↦ //    e_2_2e_1_2=e_1_2*e_2_2-e_1_2
↦ //    e_2_3e_1_2=e_1_2*e_2_3-e_1_3
↦ //    e_3_1e_1_2=e_1_2*e_3_1+e_3_2
↦ //    e_2_1e_1_3=e_1_3*e_2_1+e_2_3
↦ //    e_3_1e_1_3=e_1_3*e_3_1-e_1_1+e_3_3
↦ //    e_3_2e_1_3=e_1_3*e_3_2-e_1_2
↦ //    e_3_3e_1_3=e_1_3*e_3_3-e_1_3
↦ //    e_2_2e_2_1=e_2_1*e_2_2+e_2_1
↦ //    e_3_2e_2_1=e_2_1*e_3_2+e_3_1
↦ //    e_2_3e_2_2=e_2_2*e_2_3-e_2_3
↦ //    e_3_2e_2_2=e_2_2*e_3_2+e_3_2
↦ //    e_3_1e_2_3=e_2_3*e_3_1-e_2_1
↦ //    e_3_2e_2_3=e_2_3*e_3_2-e_2_2+e_3_3
↦ //    e_3_3e_2_3=e_2_3*e_3_3-e_2_3
↦ //    e_3_3e_3_1=e_3_1*e_3_3+e_3_1
↦ //    e_3_3e_3_2=e_3_2*e_3_3+e_3_2
ring r3 = 0,(x,y,z,Dx,Dy,Dz),dp;
def B = Weyl(); setring B; B;     // third Weyl algebra
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x y z Dx Dy Dz
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
↦ //    Dzz=z*Dz+1
ideal ff = x*Dx,x*Dy,x*Dz,y*Dx,y*Dy,y*Dz,z*Dx,z*Dy,z*Dz;
map f = A,ff;                     // f: A -> B, e(i,j) |-> x(i)D(j)
ideal J = 0;
preimageNC(A,f,J,"K");            // compute K := ker(f)
setring A;
K;
↦ K[1]=e_2_3*e_3_2-e_2_2*e_3_3-e_2_2
↦ K[2]=e_1_3*e_3_2-e_1_2*e_3_3-e_1_2
↦ K[3]=e_2_3*e_3_1-e_2_1*e_3_3-e_2_1
↦ K[4]=e_2_2*e_3_1-e_2_1*e_3_2
↦ K[5]=e_1_3*e_3_1-e_1_1*e_3_3-e_1_1
↦ K[6]=e_1_2*e_3_1-e_1_1*e_3_2
↦ K[7]=e_1_3*e_2_2-e_1_2*e_2_3+e_1_3
↦ K[8]=e_1_3*e_2_1-e_1_1*e_2_3
↦ K[9]=e_1_2*e_2_1-e_1_1*e_2_2-e_1_1
```

See also: Section 7.3.21 [preimage (plural)], page 354.

7.5.19.3 admissibleSub

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: admissibleSub(v); v intvec

Assume: The entries of v are in the range 1..nvars(basering).

Return: int, 1 if the variables indexed by the entries of v form an admissible subalgebra, 0 otherwise

Example:

```
LIB "ncpreim.lib";
ring r = 0,(e,f,h),dp;
matrix d[3][3];
d[1,2] = -h; d[1,3] = 2*e; d[2,3] = -2*f;
def A = nc_algebra(1,d);
setring A; A; // A is U(sl_2)
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
// the subalgebra generated by e,f is not admissible since [e,f]=h
admissibleSub(1..2);
↦ 0
// but the subalgebra generated by f,h is admissible since [f,h]=2f
admissibleSub(2..3);
↦ 1
```

7.5.19.4 isUpperTriangular

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: isUpperTriangular(M[,k]); M a matrix, k an optional int

Return: int, 1 if the given matrix is upper triangular,
0 otherwise.

Note: If k<>0 is given, it is checked whether M is strictly upper triangular.

Example:

```
LIB "ncpreim.lib";
ring r = 0,x,dp;
matrix M[2][3] =
0,1,2,
0,0,3;
isUpperTriangular(M);
↦ 1
isUpperTriangular(M,1);
↦ 1
M[2,2] = 4;
isUpperTriangular(M);
↦ 1
isUpperTriangular(M,1);
↦ 0
```

7.5.19.5 appendWeight2Ord

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: appendWeight2Ord(w); w an intvec

Return: ring, the basering equipped with the ordering (a(w),<), where `<` is the ordering of the basering.

Example:

```
LIB "ncpreim.lib";
ring r = 0,(a,b,x,d),Dp;
intvec w = 1,2,3,4;
def r2 = appendWeight2Ord(w); // for a commutative ring
r2;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering a
↦ //                  : names    a b x d
↦ //                  : weights  1 2 3 4
↦ //        block   2 : ordering Dp
↦ //                  : names    a b x d
↦ //        block   3 : ordering C
matrix D[4][4];
D[1,2] = 3*a;  D[1,4] = 3*x^2;  D[2,3] = -x;
D[2,4] = d;    D[3,4] = 1;
def A = nc_algebra(1,D);
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering Dp
↦ //                  : names    a b x d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=ab+3a
↦ //    da=ad+3x2
↦ //    xb=bx-x
↦ //    db=bd+d
↦ //    dx=xd+1
w = 2,1,1,1;
def B = appendWeight2Ord(w);  // for a non-commutative ring
setring B; B;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering a
↦ //                  : names    a b x d
↦ //                  : weights  2 1 1 1
↦ //        block   2 : ordering Dp
↦ //                  : names    a b x d
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    ba=ab+3a
↦ //    da=ad+3x2
↦ //    xb=bx-x
↦ //     db=bd+d
↦ //     dx=xd+1
```

7.5.19.6 elimWeight

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: elimWeight(v); v an intvec

Assume: The basering is a G-algebra.
The entries of v are in the range 1..nvars(basering) and the corresponding variables generate an admissible subalgebra.

Return: intvec, say w, such that the ordering (a(w),<), where < is any admissible global ordering, is an elimination ordering for the subalgebra generated by the variables indexed by the entries of the given intvec.

Note: If no such ordering exists, the zero intvec is returned.

Remark: Reference: (BGL), (GML)

Example:

```
LIB "ncpreim.lib";
// (Lev): Example 2
ring r = 0,(a,b,x,d),Dp;
matrix D[4][4];
D[1,2] = 3*a;  D[1,4] = 3*x^2;  D[2,3] = -x;
D[2,4] = d;    D[3,4] = 1;
def A = nc_algebra(1,D);
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering Dp
↦ //                  : names    a b x d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //     ba=ab+3a
↦ //     da=ad+3x2
↦ //     xb=bx-x
↦ //     db=bd+d
↦ //     dx=xd+1
// Since d*a-a*d = 3*x^2, any admissible ordering has to satisfy
// x^2 < a*d, while any elimination ordering for {x,d} additionally
// has to fulfil a << x and a << d.
// Hence neither a block ordering with weights
// (1,1,1,1) nor a weighted ordering with weight (0,0,1,1) will do.
intvec v = 3,4;
elimWeight(v);
↦ 0,0,1,2
```

7.5.19.7 extendedTensor

Procedure from library `ncpreim.lib` (see Section 7.5.19 [ncpreim_lib], page 554).

Usage: extendedTensor(A,I); A ring, I ideal

Return: ring, A+B (where B denotes the basering) extended with non- commutative relations between the vars of A and B, which arise from the homomorphism A -> B induced by I in the usual sense, i.e. if the vars of A are named x(i) and the vars of B y(j), then putting q(i)(j) = leadcoef(y(j)*I[i])/leadcoef(I[i]*y(j)) and r(i)(j) = y(j)*I[i] - q(i)(j)*I[i]*y(j) yields the relation y(j)*x(i) = q(i)(j)*x(i)*y(j)+r(i)(j).

Remark: Reference: (Lev)

Example:

```
LIB "ncpreim.lib";
def A = makeWeyl(2);
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) D(1) D(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    D(1)x(1)=x(1)*D(1)+1
↦ //    D(2)x(2)=x(2)*D(2)+1
def B = makeUgl(2);
setring B; B;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    e_1_1 e_1_2 e_2_1 e_2_2
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    e_1_2e_1_1=e_1_1*e_1_2-e_1_2
↦ //    e_2_1e_1_1=e_1_1*e_2_1+e_2_1
↦ //    e_2_1e_1_2=e_1_2*e_2_1-e_1_1+e_2_2
↦ //    e_2_2e_1_2=e_1_2*e_2_2-e_1_2
↦ //    e_2_2e_2_1=e_2_1*e_2_2+e_2_1
ideal I = var(1)*var(3), var(1)*var(4), var(2)*var(3), var(2)*var(4);
I;
↦ I[1]=e_1_1*e_2_1
↦ I[2]=e_1_1*e_2_2
↦ I[3]=e_1_2*e_2_1
↦ I[4]=e_1_2*e_2_2
def C = extendedTensor(A,I);
setring C; C;
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) D(1) D(2)
↦ //        block   2 : ordering dp
↦ //                  : names    e_1_1 e_1_2 e_2_1 e_2_2
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    D(1)x(1)=x(1)*D(1)+1
↦ //    e_1_1x(1)=x(1)*e_1_1-e_1_1*e_2_1
↦ //    e_1_2x(1)=x(1)*e_1_2+e_1_1^2-e_1_2*e_2_1-e_1_1*e_2_2
↦ //    e_2_1x(1)=x(1)*e_2_1+e_2_1^2
↦ //    e_2_2x(1)=x(1)*e_2_2+e_1_1*e_2_1
↦ //     D(2)x(2)=x(2)*D(2)+1
↦ //     e_1_2x(2)=x(2)*e_1_2+e_1_1*e_1_2-e_1_2*e_2_2
↦ //     e_2_1x(2)=x(2)*e_2_1-e_1_1*e_2_1+e_2_1*e_2_2
↦ //     e_1_2D(1)=D(1)*e_1_2+e_1_1*e_1_2-e_1_2*e_2_2-e_1_2
↦ //     e_2_1D(1)=D(1)*e_2_1-e_1_1*e_2_1+e_2_1*e_2_2+e_2_1
↦ //     e_1_1D(2)=D(2)*e_1_1+e_1_2*e_2_2
↦ //     e_1_2D(2)=D(2)*e_1_2+e_1_2^2
↦ //     e_2_1D(2)=D(2)*e_2_1-e_1_2*e_2_1-e_1_1*e_2_2+e_2_2^2
↦ //     e_2_2D(2)=D(2)*e_2_2-e_1_2*e_2_2
↦ //     e_1_2e_1_1=e_1_1*e_1_2-e_1_2
↦ //     e_2_1e_1_1=e_1_1*e_2_1+e_2_1
↦ //     e_2_1e_1_2=e_1_2*e_2_1-e_1_1+e_2_2
↦ //     e_2_2e_1_2=e_1_2*e_2_2-e_1_2
↦ //     e_2_2e_2_1=e_2_1*e_2_2+e_2_1
```

7.5.20 nctools_lib

Library: nctools.lib

Purpose: General tools for noncommutative algebras

Authors: Levandovskyy V., levandov@mathematik.uni-kl.de,
Lobillo, F.J., jlobillo@ugr.es,
Rabelo, C., crabelo@ugr.es,
Motsak, O., U@D, where U={motsak}, D={mathematik.uni-kl.de}

Overview: Support: DFG (Deutsche Forschungsgesellschaft) and Metodos algebraicos y efectivos en grupos cuanticos, BFM2001-3141, MCYT, Jose Gomez-Torrecillas (Main researcher).

Procedures:

7.5.20.1 Gweights

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: Gweights(r); r a ring or a square matrix

Return: intvec

Purpose: compute an appropriate weight int vector for a G-algebra, i.e., such that \foral\;i<j\;\;lm_w(d_{ij}) <_w x_i x_j.
the polynomials d_{ij} are taken from r itself, if it is of the type ring
or defined by the given square polynomial matrix

Theory: `Gweights` returns an integer vector, whose weighting should be used to redefine the G-algebra in order to get the same non-commutative structure w.r.t. a weighted ordering. If the input is a matrix and the output is the zero vector then there is not a G-algebra structure associated to these relations with respect to the given variables.
Another possibility is to use `weightedRing` to obtain directly a G-algebra with the new appropriate (weighted) ordering.

Example:

```
LIB "nctools.lib";
ring r = (0,q),(a,b,c,d),lp;
matrix C[4][4];
```

```
C[1,2]=q; C[1,3]=q; C[1,4]=1; C[2,3]=1; C[2,4]=q; C[3,4]=q;
matrix D[4][4];
D[1,4]=(q-1/q)*b*c;
def S = nc_algebra(C,D); setring S; S;
↦ // coefficients: QQ(q)
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    a b c d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=(q)*ab
↦ //    ca=(q)*ac
↦ //    da=ad+(q2-1)/(q)*bc
↦ //    db=(q)*bd
↦ //    dc=(q)*cd
Gweights(S);
↦ 2,1,1,1
def D=fetch(r,D);
Gweights(D);
↦ 2,1,1,1
```

See also: Section 7.5.20.2 [weightedRing], page 563.

7.5.20.2 weightedRing

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: weightedRing(r); r a ring

Return: ring

Purpose: equip the variables of the given ring with weights such that the relations of new ring (with weighted variables) satisfies the ordering condition for G-algebras: e.g. \forall\;i<j\;\;lm_w(d_{ij})<_w x_i x_j.

Note: activate this ring with the "setring" command

Example:

```
LIB "nctools.lib";
ring r = (0,q),(a,b,c,d),lp;
matrix C[4][4];
C[1,2]=q; C[1,3]=q; C[1,4]=1; C[2,3]=1; C[2,4]=q; C[3,4]=q;
matrix D[4][4];
D[1,4]=(q-1/q)*b*c;
def S = nc_algebra(C,D); setring S; S;
↦ // coefficients: QQ(q)
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    a b c d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=(q)*ab
↦ //    ca=(q)*ac
↦ //    da=ad+(q2-1)/(q)*bc
↦ //    db=(q)*bd
↦ //    dc=(q)*cd
```

```
def t=weightedRing(S);
setring t; t;
↦ // coefficients: QQ(q)
↦ // number of vars : 4
↦ //        block   1 : ordering M
↦ //                  : names    a b c d
↦ //                  : weights  2 1 1 1
↦ //                  : weights  0 0 0 1
↦ //                  : weights  0 0 1 0
↦ //                  : weights  0 1 0 0
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=(q)*ab
↦ //    ca=(q)*ac
↦ //    da=ad+(q2-1)/(q)*bc
↦ //    db=(q)*bd
↦ //    dc=(q)*cd
```

See also: Section 7.5.20.1 [Gweights], page 562.

7.5.20.3 ndcond

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: ndcond();

Return: ideal

Purpose: compute the non-degeneracy conditions of the basering

Note: if `printlevel > 0`, the procedure displays intermediate information (by default, `printlevel=0`)

Example:

```
LIB "nctools.lib";
ring r = (0,q1,q2),(x,y,z),dp;
matrix C[3][3];
C[1,2]=q2; C[1,3]=q1; C[2,3]=1;
matrix D[3][3];
D[1,2]=x; D[1,3]=z;
def S = nc_algebra(C,D); setring S;
S;
↦ // coefficients: QQ(q1, q2)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=(q2)*x*y+x
↦ //    zx=(q1)*x*z+z
ideal j=ndcond(); // the silent version
j;
↦ j[1]=(-q2+1)*y*z-z
printlevel=1;
ideal i=ndcond(); // the verbose version
↦ Processing degree : 1
↦ 1 . 2 . 3 .
↦ failed: (-q2+1)*y*z-z
↦ done
i;
↦ i[1]=(-q2+1)*y*z-z
```

7.5.20.4 Weyl

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: Weyl()

Return: ring

Purpose: create a Weyl algebra structure on the basering

Note: Activate this ring using the command `setring`.
Assume the number of variables of a basering is 2k. (if the number of variables is odd, an error message will be returned)
by default, the procedure treats first k variables as coordinates x_i and the last k as differentials d_i
if a non-zero optional argument is given, the procedure treats 2k variables of a basering as k pairs (x_i,d_i), i.e. variables with odd numbers are treated as coordinates and with even numbers as differentials

Example:

```
LIB "nctools.lib";
ring A1=0,(x(1..2),d(1..2)),dp;
def S=Weyl();
setring S;  S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) d(1) d(2)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    d(1)x(1)=x(1)*d(1)+1
↦ //    d(2)x(2)=x(2)*d(2)+1
kill A1,S;
ring B1=0,(x1,d1,x2,d2),dp;
def S=Weyl(1);
setring S;  S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x1 d1 x2 d2
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    d1x1=x1*d1+1
↦ //    d2x2=x2*d2+1
```

See also: Section 7.5.20.5 [makeWeyl], page 565.

7.5.20.5 makeWeyl

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: makeWeyl(n,[p]); n an integer, n>0; p an optional integer (field characteristic)

Return: ring

Purpose: create the n-th Weyl algebra over the rationals Q or F_p

Note: activate this ring with the "setring" command.
The presentation of an n-th Weyl algebra is classical: D(i)x(i)=x(i)D(i)+1,
where x(i) correspond to coordinates and D(i) to partial differentiations, i=1,...,n.
If p is not prime, the next larger prime number will be used.

Example:

```
LIB "nctools.lib";
def a = makeWeyl(3);
setring a;
a;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) D(1) D(2) D(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    D(1)x(1)=x(1)*D(1)+1
↦ //    D(2)x(2)=x(2)*D(2)+1
↦ //    D(3)x(3)=x(3)*D(3)+1
```

See also: Section 7.5.20.4 [Weyl], page 565.

7.5.20.6 makeHeisenberg

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: makeHeisenberg(n, [p,d]); int n (setting 2n+1 variables), optional int p (field characteristic), optional int d (power of h in the commutator)

Return: ring

Purpose: create the n-th Heisenberg algebra in the variables x(1),y(1),...,x(n),y(n),h over the rationals Q or F_p with the relations \forall\;i\in\{1,2,\ldots,n\}\;\;y(j)x(i) = x(i)y(j)+h^d.

Note: activate this ring with the `setring` command
If p is not prime, the next larger prime number will be used.

Example:

```
LIB "nctools.lib";
def a = makeHeisenberg(2);
setring a;   a;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering lp
↦ //                  : names    x(1) x(2) y(1) y(2) h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    y(1)x(1)=x(1)*y(1)+h
↦ //    y(2)x(2)=x(2)*y(2)+h
def H3 = makeHeisenberg(3, 7, 2);
```

```
setring H3;  H3;
↦ // coefficients: ZZ/7
↦ // number of vars : 7
↦ //        block   1 : ordering lp
↦ //                  : names    x(1) x(2) x(3) y(1) y(2) y(3) h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    y(1)x(1)=x(1)*y(1)+h^2
↦ //    y(2)x(2)=x(2)*y(2)+h^2
↦ //    y(3)x(3)=x(3)*y(3)+h^2
```

See also: Section 7.5.20.5 [makeWeyl], page 565.

7.5.20.7 Exterior

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: Exterior();

Return: qring

Purpose: create the exterior algebra of a basering

Note: activate this qring with the "setring" command

Theory: given a basering, this procedure introduces the anticommutative relations x(j)x(i)=-x(i)x(j) for all j>i,
moreover, creates a factor algebra modulo the two-sided ideal, generated by x(i)^2 for all i

Example:

```
LIB "nctools.lib";
ring R = 0,(x(1..3)),dp;
def ER = Exterior();
setring ER;
ER;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(2)x(1)=-x(1)*x(2)
↦ //    x(3)x(1)=-x(1)*x(3)
↦ //    x(3)x(2)=-x(2)*x(3)
↦ // quotient ring from ideal
↦ _[1]=x(3)^2
↦ _[2]=x(2)^2
↦ _[3]=x(1)^2
```

7.5.20.8 findimAlgebra

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: findimAlgebra(M,[r]); M a matrix, r an optional ring

Return: ring

Purpose: define a finite dimensional algebra structure on a ring

Note: the matrix M is used to define the relations x(i)*x(j) = M[i,j] in the basering (by default) or in the optional ring r.
The procedure equips the ring with the noncommutative structure.
The procedure exports the ideal (not a two-sided Groebner basis!), called `fdQuot`, for further qring definition.

Theory: finite dimensional algebra can be represented as a factor algebra of a G-algebra modulo certain two-sided ideal. The relations of a f.d. algebra are thus naturally divided into two groups: firstly, the relations on the variables of the ring, making it into G-algebra and the rest of them, which constitute the ideal which will be factored out.

Example:

```
LIB "nctools.lib";
ring r=(0,a,b),(x(1..3)),dp;
matrix S[3][3];
S[2,3]=a*x(1); S[3,2]=-b*x(1);
def A=findimAlgebra(S); setring A;
fdQuot = twostd(fdQuot);
qring Qr = fdQuot;
Qr;
↦ // coefficients: QQ(a, b)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=(-b)/(a)*x(2)*x(3)
↦ // quotient ring from ideal
↦ _[1]=x(3)^2
↦ _[2]=x(2)*x(3)+(-a)*x(1)
↦ _[3]=x(1)*x(3)
↦ _[4]=x(2)^2
↦ _[5]=x(1)*x(2)
↦ _[6]=x(1)^2
```

7.5.20.9 superCommutative

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: superCommutative([b,[e, [Q]]]);

Return: qring

Purpose: create a super-commutative algebra (as a GR-algebra) over a basering,

Note: activate this qring with the "setring" command.

Note: if b==e then the resulting ring is commutative.
By default, `b=1, e=nvars(basering), Q=0`.

Theory: given a basering, this procedure introduces the anti-commutative relations
var(j)var(i)=-var(i)var(j) for all e>=j>i>=b and creates the quotient
of the anti-commutative algebra modulo the two-sided ideal, generated by
x(b)^2, ..., x(e)^2[+ Q]

Display: If `printlevel` > 1, warning debug messages will be printed

Example:

```
LIB "nctools.lib";
ring R = 0,(x(1..4)),dp; // global!
def ER = superCommutative(); // the same as Exterior (b = 1, e = N)
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(2)x(1)=-x(1)*x(2)
↦ //    x(3)x(1)=-x(1)*x(3)
↦ //    x(4)x(1)=-x(1)*x(4)
↦ //    x(3)x(2)=-x(2)*x(3)
↦ //    x(4)x(2)=-x(2)*x(4)
↦ //    x(4)x(3)=-x(3)*x(4)
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
↦ _[4]=x(1)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 1 , 4 ].
kill R; kill ER;
ring R = 0,(x(1..4)),(lp(1), dp(3)); // global!
def ER = superCommutative(2); // b = 2, e = N
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    x(1)
↦ //        block   2 : ordering dp
↦ //                  : names    x(2) x(3) x(4)
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=-x(2)*x(3)
↦ //    x(4)x(2)=-x(2)*x(4)
↦ //    x(4)x(3)=-x(3)*x(4)
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 2 , 4 ].
kill R; kill ER;
ring R = 0,(x, y, z),(ds(1), dp(2)); // mixed!
def ER = superCommutative(2,3); // b = 2, e = 3
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering ds
↦ //                  : names    x
↦ //        block   2 : ordering dp
↦ //                  : names    y z
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    zy=-yz
↦ // quotient ring from ideal
↦ _[1]=y2
↦ _[2]=z2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 2 , 3 ].
x + 1 + z + y; // ordering on variables: y > z > 1 > x
↦ y+z+1+x
std(x - x*x*x);
↦ _[1]=x
std(ideal(x - x*x*x, x*x*z + y, z + y*x*x));
↦ _[1]=y+x2z
↦ _[2]=z+x2y
↦ _[3]=x
kill R; kill ER;
ring R = 0,(x, y, z),(ds(1), dp(2)); // mixed!
def ER = superCommutative(2, 3, ideal(x - x*x, x*x*z + y, z + y*x*x )); // b = 2, e =
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering ds
↦ //                  : names    x
↦ //        block   2 : ordering dp
↦ //                  : names    y z
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    zy=-yz
↦ // quotient ring from ideal
↦ _[1]=y+x2z
↦ _[2]=z+x2y
↦ _[3]=x
↦ _[4]=y2
↦ _[5]=z2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 2 , 3 ].
```

7.5.20.10 rightStd

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Purpose: compute a right Groebner basis of I

Return: the same type as input

Example:

```
LIB "nctools.lib";
LIB "ncalg.lib";
def A = makeUsl(2);
```

```
setring A;
ideal I = e2,f;
option(redSB);
option(redTail);
ideal LI = std(I);
LI;
↦ LI[1]=f
↦ LI[2]=h2+h
↦ LI[3]=eh+e
↦ LI[4]=e2
ideal RI = rightStd(I);
RI;
↦ RI[1]=f
↦ RI[2]=h2-h
↦ RI[3]=eh+e
↦ RI[4]=e2
```

7.5.20.11 rightNF

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: rightNF(I); v a poly/vector, M an ideal/module

Purpose: compute a right normal form of v w.r.t. M

Return: poly/vector (as of the 1st argument)

Example:

```
LIB "nctools.lib";
LIB "ncalg.lib";
ring r = 0,(x,d),dp;
def S = nc_algebra(1,1); setring S; // Weyl algebra
ideal I = x; I = std(I);
poly  p = x*d+1;
NF(p,I); // left normal form
↦ 0
rightNF(p,I); // right normal form
↦ 1
```

7.5.20.12 rightModulo

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: rightModulo(M,N); M,N are ideals/modules

Purpose: compute a right representation of the module (M+N)/N

Return: module

Assume: M,N are presentation matrices for right modules

Example:

```
LIB "nctools.lib";
LIB "ncalg.lib";
def A = makeUsl(2);
setring A;
option(redSB);
option(redTail);
ideal I = e2,f2,h2-1;
I = twostd(I);
print(matrix(I));
↦ h2-1,fh-f,eh+e,f2,2ef-h-1,e2
ideal E  = std(e);
ideal TL = e,h-1; // the result of left modulo
TL;
↦ TL[1]=e
↦ TL[2]=h-1
ideal T = rightModulo(E,I);
T = rightStd(T+I);
T = rightStd(rightNF(T,I)); // make the output canonic
T;
↦ T[1]=h+1
↦ T[2]=e
```

7.5.20.13 moduloSlim

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: moduloSlim(A,B); A,B module/matrix/ideal

Return: module

Purpose: compute `modulo` with slimgb as engine

Example:

```
LIB "nctools.lib";
LIB "ncalg.lib";
ring r; // first classical example for modulo
ideal h1=x,y,z;    ideal h2=x;
module m=moduloSlim(h1,h2);
print(m);
↦ 1,0,0, 0,
↦ 0,0,z, x,
↦ 0,x,-y,0
// now, a noncommutative example
def A = makeUsl2(); setring A; // this algebra is U(sl_2)
ideal H2 = e2,f2,h2-1; H2 = twostd(H2);
print(matrix(H2)); // print H2 in a compact form
↦ h2-1,fh-f,eh+e,f2,2ef-h-1,e2
ideal H1 = std(e);
ideal T = moduloSlim(H1,H2);
T = std( NF(std(H2+T),H2) );
T;
↦ T[1]=h-1
↦ T[2]=e
// now, a matrix example:
ring r2 = 0,(x,d), (dp);
def R = nc_algebra(1,1); setring R;
matrix M[2][2] = d, 0, 0, d*(x*d);
matrix P[2][1] = (8x+7)*d+9x, (x2+1)*d + 5*x;
module X = moduloSlim(P,M);
print(X);
```

```
↦ 5x2d2-2xd3-5xd-6d2+5,xd5+5xd3+5d4+5d2
```

7.5.20.14 ncRelations

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: ncRelations(r); r a ring

Return: list L with two elements, both elements are of type matrix:
L[1] = matrix of coefficients C,
L[2] = matrix of polynomials D

Purpose: recover the noncommutative relations via matrices C and D from a noncommutative ring

Example:

```
LIB "nctools.lib";
ring r = 0,(x,y,z),dp;
matrix C[3][3]=0,1,2,0,0,-1,0,0,0;
print(C);
↦ 0,1,2,
↦ 0,0,-1,
↦ 0,0,0
matrix D[3][3]=0,1,2y,0,0,-2x+y+1;
print(D);
↦ 0,1,2y,
↦ 0,0,-2x+y+1,
↦ 0,0,0
def S=nc_algebra(C,D);setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=xy+1
↦ //    zx=2xz+2y
↦ //    zy=-yz-2x+y+1
def l=ncRelations(S);
print (l[1]);
↦ 0,1,2,
↦ 0,0,-1,
↦ 0,0,0
print (l[2]);
↦ 0,1,2y,
↦ 0,0,-2x+y+1,
↦ 0,0,0
```

See also: Section 7.4.1 [G-algebras], page 365; Section 5.1.137 [ringlist], page 255.

7.5.20.15 isCentral

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: isCentral(p); p poly

Return: int, 1 if p commutes with all variables and 0 otherwise

Purpose: check whether p is central in a basering (that is, commutes with every generator of the ring)

Note: if `printlevel > 0`, the procedure displays intermediate information (by default, `printlevel`=0)

Example:

```
LIB "nctools.lib";
ring r=0,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z;
D[1,3]=2*x;
D[2,3]=-2*y;
def S = nc_algebra(1,D); setring S;
S; // this is U(sl_2)
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    yx=xy-z
↦ //    zx=xz+2x
↦ //    zy=yz-2y
poly c = 4*x*y+z^2-2*z;
printlevel = 0;
isCentral(c);
↦ 1
poly h = x*c;
printlevel = 1;
isCentral(h);
↦ Non-central at: y
↦ Non-central at: z
↦ 0
```

7.5.20.16 isNC

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: isNC();

Purpose: check whether a basering is commutative or not

Return: int, 1 if basering is noncommutative and 0 otherwise

Example:

```
LIB "nctools.lib";
def a = makeWeyl(2);
setring a;
isNC();
↦ 1
kill a;
ring r = 17,(x(1..7)),dp;
isNC();
↦ 0
kill r;
```

7.5.20.17 isCommutative

Procedure from library nctools.lib (see Section 7.5.20 [nctools_lib], page 562).

Usage: isCommutative();

Return: int, 1 if basering is commutative, or 0 otherwise

Purpose: check whether basering is commutative

Example:

```
LIB "nctools.lib";
ring r = 0,(x,y),dp;
isCommutative();
↦ 1
def D = Weyl(); setring D;
isCommutative();
↦ 0
setring r;
def R = nc_algebra(1,0); setring R;
isCommutative();
↦ 1
```

7.5.20.18 isWeyl

Procedure from library nctools.lib (see Section 7.5.20 [nctools_lib], page 562).

Usage: isWeyl();

Return: int, 1 if basering is a Weyl algebra, or 0 otherwise

Purpose: check whether basering is a Weyl algebra

Example:

```
LIB "nctools.lib";
ring r = 0,(a,b,c,d),dp;
isWeyl();
↦ 0
def D = Weyl(1); setring D; //make from r a Weyl algebra
b*a;
↦ ab+1
isWeyl();
↦ 1
ring t = 0,(Dx,x,y,Dy),dp;
matrix M[4][4]; M[1,2]=-1; M[3,4]=1;
def T = nc_algebra(1,M); setring T;
isWeyl();
↦ 1
```

7.5.20.19 UpOneMatrix

Procedure from library nctools.lib (see Section 7.5.20 [nctools_lib], page 562).

Usage: UpOneMatrix(n); n an integer

Return: intmat

Purpose: compute an n x n matrix with 1's in the whole upper triangle

Note: helpful for setting noncommutative algebras with complicated coefficient matrices

Example:

```
LIB "nctools.lib";
ring   r = (0,q),(x,y,z),dp;
matrix C = UpOneMatrix(3);
C[1,3]   = q;
print(C);
↦ 0,1,(q),
↦ 0,0,1,
↦ 0,0,0
def S = nc_algebra(C,0); setring S;
S;
↦ // coefficients: QQ(q)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    zx=(q)*xz
```

7.5.20.20 AltVarStart

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: AltVarStart();

Return: int

Purpose: returns the number of the first alternating variable of basering

Note: basering should be a super-commutative algebra constructed by the procedure `superCommutative`, emits an error otherwise

Example:

```
LIB "nctools.lib";
ring R = 0,(x(1..4)),dp; // global!
def ER = superCommutative(2); // (b = 2, e = N)
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=-x(2)*x(3)
↦ //    x(4)x(2)=-x(2)*x(4)
↦ //    x(4)x(3)=-x(3)*x(4)
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 2 , 4 ].
setring R;
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
```

```
↦    ? SCA rings are factors by (at least) squares!
↦    ? leaving nctools.lib::AltVarStart (1133)
kill R, ER;
//////////////////////////////////////////////////////////////////
ring R = 2,(x(1..4)),dp; // the same in char. = 2!
def ER = superCommutative(2); // (b = 2, e = N)
setring ER; ER;
↦ // coefficients: ZZ/2
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 4 , 4 ].
setring R;
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦    ? SCA rings are factors by (at least) squares!
↦    ? leaving nctools.lib::AltVarStart (1133)
```

7.5.20.21 AltVarEnd

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: AltVarStart();

Return: int

Purpose: returns the number of the last alternating variable of basering

Note: basering should be a super-commutative algebra constructed by the procedure `superCommutative`, emits an error otherwise

Example:

```
LIB "nctools.lib";
ring R = 0,(x(1..4)),dp; // global!
def ER = superCommutative(2); // (b = 2, e = N)
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=-x(2)*x(3)
↦ //    x(4)x(2)=-x(2)*x(4)
↦ //    x(4)x(3)=-x(3)*x(4)
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
```

```
↦ Alternating variables: [ 2 , 4 ].
setring R;
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦    ? SCA rings are factors by (at least) squares!
↦    ? leaving nctools.lib::AltVarStart (1133)
kill R, ER;
//////////////////////////////////////////////////////////////////
ring R = 2,(x(1..4)),dp; // the same in char. = 2!
def ER = superCommutative(2); // (b = 2, e = N)
setring ER; ER;
↦ // coefficients: ZZ/2
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦ Alternating variables: [ 4 , 4 ].
setring R;
"Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
↦    ? SCA rings are factors by (at least) squares!
↦    ? leaving nctools.lib::AltVarStart (1133)
```

7.5.20.22 IsSCA

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: IsSCA();

Return: int

Purpose: returns 1 if basering is a super-commutative algebra and 0 otherwise

Example:

```
LIB "nctools.lib";
/////////////////////////////////////////////////////////////////////
ring R = 0,(x(1..4)),dp; // commutative
if(IsSCA())
{ "Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "]."; }
else
{ "Not a super-commutative algebra!!!"; }
↦ Not a super-commutative algebra!!!
kill R;
/////////////////////////////////////////////////////////////////////
ring R = 0,(x(1..4)),dp;
def S = nc_algebra(1, 0); setring S; S; // still commutative!
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
```

```
if(IsSCA())
{ "Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "]."; }
else
{ "Not a super-commutative algebra!!!"; }
↦ Not a super-commutative algebra!!!
kill R, S;
/////////////////////////////////////////////////////////////////////
ring R = 0,(x(1..4)),dp;
list CurrRing = ringlist(R);
def ER = ring(CurrRing);
setring ER; // R;
matrix E = UpOneMatrix(nvars(R));
int i, j; int b = 2; int e = 3;
for ( i = b; i < e; i++ )
{
for ( j = i+1; j <= e; j++ )
{
E[i, j] = -1;
}
}
def S = nc_algebra(E,0); setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=-x(2)*x(3)
if(IsSCA())
{ "Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "]."; }
else
{ "Not a super-commutative algebra!!!"; }
↦ Not a super-commutative algebra!!!
kill R, ER, S;
/////////////////////////////////////////////////////////////////////
ring R = 0,(x(1..4)),dp;
def ER = superCommutative(2); // (b = 2, e = N)
setring ER; ER;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    x(3)x(2)=-x(2)*x(3)
↦ //    x(4)x(2)=-x(2)*x(4)
↦ //    x(4)x(3)=-x(3)*x(4)
↦ // quotient ring from ideal
↦ _[1]=x(4)^2
↦ _[2]=x(3)^2
↦ _[3]=x(2)^2
if(IsSCA())
{ "This is a SCA! Alternating variables: [", AltVarStart(), ",", AltVarEnd(), "].";
```

```
↦ This is a SCA! Alternating variables: [ 2 , 4 ].
else
{ "Not a super-commutative algebra!!!"; }
kill R, ER;
```

7.5.20.23 makeModElimRing

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: makeModElimRing(L); L a list

Return: ring

Purpose: create a copy of a given ring equipped with the elimination ordering for module components `(c,<)`

Note: usually the list argument contains a ring to work with

Example:

```
LIB "nctools.lib";
ring r1 = 0,(x,y,z),(C,Dp);
def r2 = makeModElimRing(r1); setring r2; r2;   kill r2;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering c
↦ //        block   2 : ordering Dp
↦ //                  : names    x y z
ring r3 = 0,(z,t),(wp(2,3),c);
def r2 = makeModElimRing(r3); setring r2; r2; kill r2;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering c
↦ //        block   2 : ordering wp
↦ //                  : names    z t
↦ //                  : weights  2 3
ring r4 = 0,(z,t,u,w),(a(1,2),C,wp(2,3,4,5));
def r2 = makeModElimRing(r4); setring r2; r2;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering c
↦ //        block   2 : ordering a
↦ //                  : names    z t
↦ //                  : weights  1 2
↦ //        block   3 : ordering wp
↦ //                  : names    z t u w
↦ //                  : weights  2 3 4 5
```

7.5.20.24 embedMat

Procedure from library `nctools.lib` (see Section 7.5.20 [nctools_lib], page 562).

Usage: embedMat(A,m,n); A,B matrix/module

Return: matrix

Purpose: embed A in the left upper corner of mxn matrix

Example:

```
LIB "nctools.lib";
ring r = 0,(a,b,c,d),dp;
matrix M[2][3]; M[1,1]=a; M[1,2]=b;M[2,2]=d;M[1,3]=c;
print(M);
↦ a,b,c,
↦ 0,d,0
print(embedMat(M,3,4));
↦ a,b,c,0,
↦ 0,d,0,0,
↦ 0,0,0,0
matrix N = M; N[2,2]=0;
print(embedMat(N,3,4));
↦ a,b,c,0,
↦ 0,0,0,0,
↦ 0,0,0,0
```

7.5.21 olga_lib

Library: olga.lib

Purpose: Ore-localization in G-Algebras

Author: Johannes Hoffmann, email: johannes.hoffmann at math.rwth-aachen.de

Overview: Let A be a G-algebra.

Current localization types:
Type 0: monoidal
- represented by a list of polys g_1,...,g_k that have to be contained in a commutative polynomial subring of A generated by a subset of the variables of A
Type 1: geometric
- only for algebras with an even number of variables where the first half induces a commutative polynomial subring B of A
- represented by an ideal p, which has to be a prime ideal in B Type 2: rational
- represented by an intvec v = [i_1,...,i_k] in the range 1..nvars(basering)

Localization data is an int specifying the type and a def with the corresponding information.

A fraction is represented as a vector with four entries: [s,r,p,t] Here, s^{-1}r is the left fraction representation, pt^{-1} is the right one. If s or t is zero, it means that the corresponding representation is not set. If both are zero, the fraction is not valid.

A detailed description along with further examples can be found in our paper:
Johannes Hoffmann, Viktor Levandovskyy:
Constructive Arithmetics in Ore Localizations of Domains
https://arxiv.org/abs/1712.01773

Procedures:

7.5.21.1 locStatus

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: locStatus(locType, locData), int locType, list/vector/intvec locData

Purpose: determine the status of a set of localization data

Assume:

Return: list

Note: - the first entry is 0 or 1, depending whether the input represents a valid localization
- the second entry is a string with a status/error message

Example:

```
LIB "olga.lib";
locStatus(42, list(1));
↦ [1]:
↦    0
↦ [2]:
↦    invalid localization: type is 42, valid types are:
↦ 0 for a monoidal localization
↦ 1 for a geometric localization
↦ 2 for a rational localization
def undef;
locStatus(0, undef);
↦ [1]:
↦    0
↦ [2]:
↦    uninitialized or invalid localization: locData has to be defined
string s;
locStatus(0, s);
↦ [1]:
↦    0
↦ [2]:
↦    for a monoidal localization, locData has to be of type list, but is of\
    type string
list L;
locStatus(0, L);
↦ [1]:
↦    0
↦ [2]:
↦    for a monoidal localization, locData has to be a non-empty list
L = s;
print(L);
↦ [1]:
↦
locStatus(0, L);
↦ [1]:
↦    0
↦ [2]:
↦    for a monoidal localization, locData has to be a list of polys, ints o\
   r numbers, but entry 1 is , which is of type string
ring w = 0,(x,Dx,y,Dy),dp;
def W = Weyl(1);
setring W;
W;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x Dx y Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
locStatus(0, list(x, Dx));
↦ [1]:
↦    0
↦ [2]:
↦    for a monoidal localization, the variables occurring in the polys in l\
   ocData have to induce a commutative polynomial subring of basering
ring R;
setring R;
R;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
locStatus(1, s);
↦ [1]:
↦    0
↦ [2]:
↦    for a geometric localization, basering has to have an even number of v\
   ariables
setring W;
locStatus(1, s);
↦ [1]:
↦    0
↦ [2]:
↦    for a geometric localization, the first half of the variables of baser\
   ing has to induce a commutative polynomial subring of basering
ring t = 0,(x,y,Dx,Dy),dp;
def T = Weyl();
setring T;
T;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
locStatus(1, s);
↦ [1]:
↦    0
↦ [2]:
↦    for a geometric localization, locData has to be of type ideal, but is \
   of type string
locStatus(1, ideal(Dx));
↦ [1]:
↦    0
↦ [2]:
```

```
↦    for a geometric localization, locData has to be an ideal generated by \
   polynomials containing only variables from the first half of the variable\
   s
locStatus(2, s);
↦ [1]:
↦    0
↦ [2]:
↦    for a rational localization, locData has to be of type intvec, but is \
   of type string
intvec v;
locStatus(2, v);
↦ [1]:
↦    0
↦ [2]:
↦    for a rational localization, locData has to be a non-zero intvec
locStatus(2, intvec(1,2));
↦ [1]:
↦    1
↦ [2]:
↦    valid localization
```

7.5.21.2 testLocData

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: testLocData(locType, locData), int locType, list/vector/intvec locData

Purpose: test if the given data specifies a denominator set wrt. the checks from locStatus

Assume:

Return: nothing

Note: throws error if checks were not successful

Example:

```
LIB "olga.lib";
ring R; setring R;
testLocData(0, list(1)); // correct localization, no error
testLocData(42, list(1)); // incorrect localization, results in error
↦    ? invalid localization: type is 42, valid types are:
↦ 0 for a monoidal localization
↦ 1 for a geometric localization
↦ 2 for a rational localization
↦    ? leaving olga.lib::testLocData (0)
```

7.5.21.3 isInS

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: isInS(p, locType, locData(, override)), poly p, int locType, list/vector/intvec locData(, int override)

Purpose: determine if a polynomial is in a denominator set

Assume:

Return: int

Note: - returns 0 or 1, depending whether p is in the denominator set specified by locType and locData
- if override is set, will not normalize locData (use with care)

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x^2*y+x+2;
poly g2 = y^3+x*y;
list L = g1,g2;
poly g = g1^2*g2;
poly f = g-1;
isInS(g, 0, L); // g is in the denominator set
↦ 1
isInS(f, 0, L); // f is NOT in the denominator set
↦ 0
// geometric localization
ideal p = x-1, y-3;
g = x^2+y-3;
f = (x-1)*g;
isInS(g, 1, p); // g is in the denominator set
↦ 1
isInS(f, 1, p); // f is NOT in the denominator set
↦ 0
// rational localization
intvec v = 2;
g = y^5+17*y^2-4;
f = x*y;
isInS(g, 2, v); // g is in the denominator set
↦ 1
isInS(f, 2, v); // f is NOT in the denominator set
↦ 0
```

7.5.21.4 fracStatus

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: fracStatus(frac, locType, locData), vector frac, int locType, list/intvec/vector locData

Purpose: determine if the given vector is a representation of a fraction in the specified localization

Assume:

Return: list

Note: - the first entry is 0 or 1, depending whether the input is valid - the second entry is a string with a status message

Example:

```
LIB "olga.lib";
ring r = QQ[x,y,Dx,Dy];
def R = Weyl();
setring R;
fracStatus([1,0,0,0], 42, list(1));
↦ [1]:
↦    0
↦ [2]:
↦    invalid localization in fraction: gen(1)
↦       invalid localization: type is 42, valid types are:
↦ 0 for a monoidal localization
↦ 1 for a geometric localization
↦ 2 for a rational localization
list L = x;
fracStatus([0,7,x,0], 0, L);
↦ [1]:
↦    0
↦ [2]:
↦    vector is not a valid fraction: no denominator specified in x*gen(3)+7\
   *gen(2)
fracStatus([Dx,Dy,0,0], 0, L);
↦ [1]:
↦    0
↦ [2]:
↦    the left denominator Dx of fraction Dx*gen(1)+Dy*gen(2) is not in the \
   denominator set of type 0 given by x
fracStatus([0,0,Dx,Dy], 0, L);
↦ [1]:
↦    0
↦ [2]:
↦    the right denominator Dy of fraction Dx*gen(3)+Dy*gen(4) is not in the\
    denominator set of type 0 given by x
fracStatus([x,Dx,Dy,x], 0, L);
↦ [1]:
↦    0
↦ [2]:
↦    left and right representation are not equal in:x*gen(4)+x*gen(1)+Dx*ge\
   n(2)+Dy*gen(3)
fracStatus([x,Dx,x*Dx+2,x^2], 0, L);
↦ [1]:
↦    1
↦ [2]:
↦    valid fraction
```

7.5.21.5 testFraction

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: testFraction(frac, locType, locData), vector frac, int locType, list/intvec/vector locData

Purpose: test if the given vector is a representation of a fraction in the specified localization wrt. the checks from fracStatus

Assume:

Return: nothing

Note: throws error if checks were not successful

Example:

```
LIB "olga.lib";
ring r = QQ[x,y,Dx,Dy];
def R = Weyl();
setring R;
list L = x;
vector frac = [x,Dx,x*Dx+2,x^2];
testFraction(frac, 0, L); // correct localization, no error
frac = [x,Dx,x*Dx,x^2];
testFraction(frac, 0, L); // incorrect localization, results in error
↦    ? left and right representation are not equal in:x^2*gen(4)+x*Dx*gen(3\
   )+x*gen(1)+Dx*gen(2)
↦    ? leaving olga.lib::testFraction (0)
```

7.5.21.6 leftOre

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: leftOre(s, r, locType, locData), poly s, r, int locType, list/vector/intvec locData

Purpose: compute left Ore data for a given tuple (s,r)

Assume: s is in the denominator set determined via locType and locData

Return: list

Note: - the first entry of the list is a vector [ts,tr] such that ts*r=tr*s - the second entry of the list is a description of all choices for ts

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// left Ore
// monoidal localization
poly g1 = x+3;
poly g2 = x*y;
```

```
list L = g1,g2;
poly g = g1^2*g2;
poly f = Dx;
list rm = leftOre(g, f, 0, L);
print(rm[1]);
↦ [x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4,x^5*y^3*Dx+6*x^4*y^\
   3*Dx-3*x^4*y^3+9*x^3*y^3*Dx-12*x^3*y^3-9*x^2*y^3]
rm[2];
↦ _[1]=x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4
rm[1][2]*g-rm[1][1]*f;
↦ 0
// geometric localization
ideal p = x-1, y-3;
f = Dx;
g = x^2+y;
list rg = leftOre(g, f, 1, p);
print(rg[1]);
↦ [x^4+2*x^2*y+y^2,x^2*Dx+y*Dx-2*x]
rg[2];
↦ _[1]=x^4+2*x^2*y+y^2
rg[1][2]*g-rg[1][1]*f;
↦ 0
// rational localization
intvec rat = 1;
f = Dx+Dy;
g = x;
list rr = leftOre(g, f, 2, rat);
print(rr[1]);
↦ [x^2,x*Dx+x*Dy-1]
rr[2];
↦ _[1]=x^2
rr[1][2]*g-rr[1][1]*f;
↦ 0
```

7.5.21.7 rightOre

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: rightOre(s, r, locType, locData), poly s, r, int locType, list/vector/intvec locData

Purpose: compute right Ore data for a given tuple (s,r)

Assume: s is in the denominator set determined via locType and locData

Return: list

Note: - the first entry of the list is a vector [ts,tr] such that r*ts=s*tr - the second entry of the list is a description of all choices for ts

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x+3;
poly g2 = x*y;
list L = g1,g2;
poly g = g1^2*g2;
poly f = Dx;
list rm = rightOre(g, f, 0, L);
print(rm[1]);
↦ [x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4,x^5*y^3*Dx+6*x^4*y^\
   3*Dx+8*x^4*y^3+9*x^3*y^3*Dx+36*x^3*y^3+36*x^2*y^3]
rm[2];
↦ _[1]=x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4
g*rm[1][2]-f*rm[1][1];
↦ 0
// geometric localization
ideal p = x-1, y-3;
f = Dx;
g = x^2+y;
list rg = rightOre(g, f, 1, p);
print(rg[1]);
↦ [x^4+2*x^2*y+y^2,x^2*Dx+y*Dx+4*x]
rg[2];
↦ _[1]=x^4+2*x^2*y+y^2
g*rg[1][2]-f*rg[1][1];
↦ 0
// rational localization
intvec rat = 1;
f = Dx+Dy;
g = x;
list rr = rightOre(g, f, 2, rat);
print(rr[1]);
↦ [x^2,x*Dx+x*Dy+2]
rr[2];
↦ _[1]=x^2
g*rr[1][2]-f*rr[1][1];
↦ 0
```

7.5.21.8 convertRightToLeftFraction

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: convertRightToLeftFraction(frac, locType, locData),
vector frac, int locType, list/vector/intvec locData

Purpose: determine a left fraction representation of a given fraction

Assume:

Return: vector

Note: - the returned vector contains a repr. of frac as a left fraction - if the left representation of frac is already specified, frac will be returned.

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x+3;
poly g2 = x*y;
list L = g1,g2;
poly g = g1^2*g2;
poly f = Dx;
vector fracm = [0,0,f,g];
vector rm = convertRightToLeftFraction(fracm, 0, L);
print(rm);
↦ [x^8*y^4+12*x^7*y^4+54*x^6*y^4+108*x^5*y^4+81*x^4*y^4,x^5*y^3*Dx+6*x^4*y^\
   3*Dx-3*x^4*y^3+9*x^3*y^3*Dx-12*x^3*y^3-9*x^2*y^3,Dx,x^3*y+6*x^2*y+9*x*y]
rm[2]*g-rm[1]*f;
↦ 0
// geometric localization
ideal p = x-1, y-3;
f = Dx;
g = x^2+y;
vector fracg = [0,0,f,g];
vector rg = convertRightToLeftFraction(fracg, 1, p);
print(rg);
↦ [x^4+2*x^2*y+y^2,x^2*Dx+y*Dx-2*x,Dx,x^2+y]
rg[2]*g-rg[1]*f;
↦ 0
// rational localization
intvec rat = 1;
f = Dx+Dy;
g = x;
vector fracr = [0,0,f,g];
vector rr = convertRightToLeftFraction(fracr, 2, rat);
print(rr);
↦ [x^2,x*Dx+x*Dy-1,Dx+Dy,x]
rr[2]*g-rr[1]*f;
↦ 0
```

7.5.21.9 convertLeftToRightFraction

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: convertLeftToRightFraction(frac, locType, locData), vector frac, int locType, list/vector/intvec locData

Purpose: determine a right fraction representation of a given fraction

Assume:

Return: vector

Note: - the returned vector contains a repr. of frac as a right fraction, - if the right representation of frac is already specified, frac will be returned.

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g = x;
poly f = Dx;
vector fracm = [g,f,0,0];
list L = g;
vector rm = convertLeftToRightFraction(fracm, 0, L);
print(rm);
↦ [x,Dx,x*Dx+2,x^2]
f*rm[4]-g*rm[3];
↦ 0
// geometric localization
g = x+y;
f = Dx+Dy;
vector fracg = [g,f,0,0];
ideal p = x-1, y-3;
vector rg = convertLeftToRightFraction(fracg, 1, p);
print(rg);
↦ [x+y,Dx+Dy,x*Dx+y*Dx+x*Dy+y*Dy+4,x^2+2*x*y+y^2]
f*rg[4]-g*rg[3];
↦ 0
// rational localization
intvec rat = 1;
f = Dx+Dy;
g = x;
vector fracr = [g,f,0,0];
vector rr = convertLeftToRightFraction(fracr, 2, rat);
print(rr);
↦ [x,Dx+Dy,x*Dx+x*Dy+2,x^2]
f*rr[4]-g*rr[3];
↦ 0
```

7.5.21.10 addLeftFractions

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: addLeftFractions(a, b, locType, locData(, override)),
vector a, b, int locType, list/vector/intvec locData(, int override)

Purpose: add two left fractions in the specified localization

Assume:

Return: vector

Note: the returned vector is the sum of a and b as fractions in the localization specified by locType and locData.

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x+3;
poly g2 = x*y+y;
list L = g1,g2;
poly s1 = g1;
poly s2 = g2;
poly r1 = Dx;
poly r2 = Dy;
vector frac1 = [s1,r1,0,0];
vector frac2 = [s2,r2,0,0];
vector rm = addLeftFractions(frac1, frac2, 0, L);
print(rm);
↦ [x^2*y+4*x*y+3*y,x*y*Dx+y*Dx+x*Dy+3*Dy]
// geometric localization
ideal p = x-1, y-3;
vector rg = addLeftFractions(frac1, frac2, 1, p);
print(rg);
↦ [x^2*y+4*x*y+3*y,x*y*Dx+y*Dx+x*Dy+3*Dy]
// rational localization
intvec v = 2;
s1 = y^2+y+1;
s2 = y-2;
r1 = Dx;
r2 = Dy;
frac1 = [s1,r1,0,0];
frac2 = [s2,r2,0,0];
vector rr = addLeftFractions(frac1, frac2, 2, v);
```

```
print(rr);
↦ [y^3-y^2-y-2,y^2*Dy+y*Dx+y*Dy-2*Dx+Dy]
```

7.5.21.11 multiplyLeftFractions

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: multiplyLeftFractions(a, b, locType, locData(, override)), vector a, b, int locType, list/vector/intvec locData, int override

Purpose: multiply two left fractions in the specified localization

Assume:

Return: vector

Note: the returned vector is the product of a and b as fractions in the localization specified by locType and locData.

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal localization
poly g1 = x+3;
poly g2 = x*y+y;
list L = g1,g2;
poly s1 = g1;
poly s2 = g2;
poly r1 = Dx;
poly r2 = Dy;
vector frac1 = [s1,r1,0,0];
vector frac2 = [s2,r2,0,0];
vector rm = multiplyLeftFractions(frac1, frac2, 0, L);
print(rm);
↦ [x^3*y^2+5*x^2*y^2+7*x*y^2+3*y^2,x*y*Dx*Dy+y*Dx*Dy-y*Dy]
// geometric localization
ideal p = x-1, y-3;
vector rg = multiplyLeftFractions(frac1, frac2, 1, p);
print(rg);
↦ [x^3*y+5*x^2*y+7*x*y+3*y,x*Dx*Dy+Dx*Dy-Dy]
// rational localization
intvec v = 2;
s1 = y^2+y+1;
s2 = y-2;
r1 = Dx;
r2 = Dy;
```

```
frac1 = [s1,r1,0,0];
frac2 = [s2,r2,0,0];
vector rr1 = multiplyLeftFractions(frac1, frac2, 2, v);
print(rr1);
↦ [y^3-y^2-y-2,Dx*Dy]
vector rr2 = multiplyLeftFractions(frac2, frac1, 2, v);
print(rr2);
↦ [y^5-y^3-4*y^2-3*y-2,y^2*Dx*Dy+y*Dx*Dy-2*y*Dx+Dx*Dy-Dx]
areEqualLeftFractions(rr1, rr2, 2, v);
↦ 0
```

7.5.21.12 areEqualLeftFractions

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: areEqualLeftFractions(a, b, locType, locData), vector a, b, int locType, list/vector/intvec locData

Purpose: check if two given fractions are equal

Assume:

Return: int

Note: returns 1 or 0, depending whether a=b as fractions in the localization specified by locType and locData

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
// monoidal
poly g1 = x*y+3;
poly g2 = y^3;
list L = g1,g2;
poly s1 = g1;
poly s2 = s1*g2;
poly s3 = s2;
poly r1 = Dx;
poly r2 = g2*r1;
poly r3 = s1*r1+3;
vector fracm1 = [s1,r1,0,0];
vector fracm2 = [s2,r2,0,0];
vector fracm3 = [s3,r3,0,0];
areEqualLeftFractions(fracm1, fracm2, 0, L);
↦ 1
areEqualLeftFractions(fracm1, fracm3, 0, L);
```

```
↦ 0
areEqualLeftFractions(fracm2, fracm3, 0, L);
↦ 0
```

7.5.21.13 isInvertibleLeftFraction

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: isInvertibleLeftFraction(frac, locType, locData), vector frac, int locType, list/vector/intvec locData

Purpose: check if a fraction is invertible in the specified localization

Assume:

Return: int

Note: - returns 1, if the numerator of frac is in the denominator set, - returns 0, otherwise (NOTE: this does NOT mean that the fraction is not invertible, it just means it could not be determined by the method above).

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
poly g1 = x+3;
poly g2 = x*y;
list L = g1,g2;
vector frac = [g1*g2, 17, 0, 0];
isInvertibleLeftFraction(frac, 0, L);
↦ 1
ideal p = x-1, y;
frac = [g1, x, 0, 0];
isInvertibleLeftFraction(frac, 1, p);
↦ 1
intvec rat = 1,2;
frac = [g1*g2, Dx, 0, 0];
isInvertibleLeftFraction(frac, 2, rat);
↦ 0
```

7.5.21.14 invertLeftFraction

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: invertLeftFraction(frac, locType, locData), vector frac, int locType, list/vector/intvec locData

Purpose: invert a fraction in the specified localization

Assume: frac is invertible in the loc. specified by locType and locData

Return: vector

Note: - returns the multiplicative inverse of frac in the localization specified by locType and locData,
- throws error if frac is not invertible (NOTE: this does NOT mean that the fraction is not invertible, it just means it could not be determined by the method listed above).

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
poly g1 = x+3;
poly g2 = x*y;
list L = g1,g2;
vector frac = [g1*g2, 17, 0, 0];
print(invertLeftFraction(frac, 0, L));
↦ [17,x^2*y+3*x*y]
ideal p = x-1, y;
frac = [g1, x, 0, 0];
print(invertLeftFraction(frac, 1, p));
↦ [x,x+3]
intvec rat = 1,2;
frac = [g1*g2, y, 0, 0];
print(invertLeftFraction(frac, 2, rat));
↦ [y,x^2*y+3*x*y]
```

7.5.21.15 isZeroFraction

Procedure from library `olga.lib` (see Section 7.5.21 [olga_lib], page 581).

Usage: isZeroFraction(frac), vector frac

Purpose: determine if the vector frac represents zero

Assume: frac is a valid fraction

Return: int

Note: returns 1, if frac == 0; 0 otherwise

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
isZeroFraction([42,0,0,0]);
↦ 1
isZeroFraction([0,0,Dx,3]);
↦ 0
isZeroFraction([1,1,1,1]);
↦ 0
```

7.5.21.16 isOneFraction

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: isOneFraction(frac), vector frac

Purpose: determine if the vector frac represents one

Assume: frac is a valid fraction

Return: int

Note: 1, if frac == 1; 0 otherwise

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl();
setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y Dx Dy
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    Dxx=x*Dx+1
↦ //    Dyy=y*Dy+1
isOneFraction([42,42,0,0]);
↦ 1
isOneFraction([0,0,Dx,3]);
↦ 0
isOneFraction([1,0,0,1]);
↦ 0
```

7.5.21.17 normalizeMonoidal

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: normalizeMonoidal(L), list L

Purpose: compute a normal form of monoidal localization data

Return: list

Note: given a list of polys, returns a list of all unique factors appearing in the given polys

Example:

```
LIB "olga.lib";
ring R = 0,(x,y,Dx,Dy),dp;
def S = Weyl(); setring S;
list L = x^2*y^3, (x+1)*(x*y-3*y^2+1);
L = normalizeMonoidal(L);
print(L);
↦ [1]:
↦    x*y-3*y^2+1
↦ [2]:
↦    x+1
↦ [3]:
↦    x
↦ [4]:
↦    y
```

7.5.21.18 normalizeRational

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: normalizeRational(v), intvec v

Purpose: compute a normal form of rational localization data

Return: intvec

Note: purges double entries and sorts ascendingly

Example:

```
LIB "olga.lib";
ring R; setring R;
intvec v = 9,5,9,3,1,5;
v = normalizeRational(v);
v;
↦ 1,3,5,9
```

7.5.21.19 testOlga

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: testOlga()

Purpose: execute a series of internal testing procedures

Return: nothing

Note:

7.5.21.20 testOlgaExamples

Procedure from library olga.lib (see Section 7.5.21 [olga_lib], page 581).

Usage: testOlgaExamples()

Purpose: execute the examples of all procedures in this library

Return: nothing

Note:

7.5.22 perron_lib

Library: perron.lib

Purpose: computation of algebraic dependences

Author: Oleksandr Motsak U@D, where U={motsak}, D={mathematik.uni-kl.de}

Procedures:

7.5.22.1 perron

Procedure from library `perron.lib` (see Section 7.5.22 [perron_lib], page 599).

Usage: perron(L [, D])

Return: commutative ring with ideal ‘Relations‘

Purpose: computes polynomial relations (’Relations’) between pairwise commuting polynomials of L [, up to a given degree bound D]

Note: the implementation was partially inspired by the Perron’s theorem.

Example:

```
LIB "perron.lib";
int p = 3;
ring AA = p,(x,y,z),dp;
matrix D[3][3]=0;
D[1,2]=-z; D[1,3]=2*x; D[2,3]=-2*y;
def A = nc_algebra(1,D); setring A; // this algebra is U(sl_2)
ideal I = x^p, y^p, z^p-z, 4*x*y+z^2-2*z; // the center
def RA = perron( I, p );
setring RA;
RA;
↦ // coefficients: ZZ/3
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    F(1) F(2) F(3) F(4)
↦ //        block   2 : ordering C
Relations; // it was exported from perron to be in the returned ring.
↦ Relations[1]=F(4)^3-F(1)*F(2)-F(3)^2+F(4)^2
// perron can be also used in a commutative case, for example:
ring B = 0,(x,y,z),dp;
ideal J = xy+z2, z2+y2, x2y2-2xy3+y4;
def RB = perron(J);
setring RB;
Relations;
↦ Relations[1]=F(1)^2-2*F(1)*F(2)+F(2)^2-F(3)
// one more test:
setring A;
map T=RA,I;
T(Relations);  // should be zero
↦ _[1]=0
```

7.5.23 purityfiltration_lib

Status: experimental

Library: purityfiltration.lib

Purpose: Algorithms for computing a purity filtration of a given module

Authors: Christian Schilli, christian.schilli@rwth-aachen.de
Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview: Purity is a notion with several meanings. In our context it is equidimensionality
of a module (that is all M is pure iff any nonzero submodule of N has the same dimension as N).
Notably, one should define purity with respect to a given dimension function. In the context
of this library the corresponding function is the homological grade number j_A(M) of a module M over
an K-algebra A. j_A(M) is the minimal integer k, such that Ext^k_A(M,A) != 0.

References:

[AQ] Alban Quadrat: Grade filtration of linear functional systems, INRIA Report 7769 (2010), to appear in Acta Applicanda Mathematica.
[B93] Jan-Erik Bjoerk: Analytic D-modules and applications, Kluwer Acad. Publ., 1993.
[MB10] Mohamed Barakat: Purity Filtration and the Fine Structure of Autonomy. Proc. MTNS, 2010.

Procedures:

7.5.23.1 projectiveDimension

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: projectiveDimension(R,i,j), R matrix representing the Modul M=coker(R)
int i, with i=0 or i=1, j a natural number

Return: list T, a projective resolution of M and its projective dimension

Purpose: if i=0 (and by default), T[1] gives a shortest left resolution of M=D^p/D^q(R^t) and T[2] the left projective dimension of M
if i=1, T[1] gives a shortest right resolution of M=D^p/RD^q and T[2] the right projective dimension of M
in both cases T[1][j] is the (j-1)-th syzygy module of M

Note: The algorithm is due to A. Quadrat, D. Robertz, Computation of bases of free modules over the Weyl algebras, J.Symb.Comp. 42, 2007.

Example:

```
LIB "purityfiltration.lib";
// commutative example
ring D = 0,(x,y,z),dp;
matrix R[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
// compute a left resolution of M=D^4/D^6*R
list T=projectiveDimension(transpose(R),0);
// so we have the left projective dimension
T[2];
↦ 3
//we could also compute a right resolution of M=D^6/RD^4
list T1=projectiveDimension(R,1);
// and we have right projective dimension
T1[2];
↦ 1
// check, that a syzygy matrix of R has left inverse:
print(leftInverse(syz(R)));
↦ 0,0,1,0
// so lpd(M) must be 1.
// Non-commutative example
ring D1 = 0,(x1,x2,x3,d1,d2,d3),dp;
def S=Weyl();  setring S;
matrix R[3][3]=
1/2*x2*d1, x2*d2+1, x2*d3+1/2*d1,
-1/2*x2*d2-3/2,0,1/2*d2,
-d1-1/2*x2*d3,-d2,-1/2*d3;
list T=projectiveDimension(R,0);
// left projective dimension of coker(R) is
T[2];
↦ 1
list T1=projectiveDimension(R,1);
// both modules have the same projective dimension, but different resolutions, becaus
print(T[1][1]);
↦ 1/2*x2*d1,   -1/2*x2*d2-3/2,-1/2*x2*d3-d1,
↦ x2*d2+1,     0,             -d2,
↦ x2*d3+1/2*d1,1/2*d2,        -1/2*d3
// not the same as
print(transpose(T1[1][1]));
↦ 1/2*x2*d1,   -1/2*x2*d2-3/2,-1/2*x2*d3-d1,-1/2*x2,
↦ x2*d2+1,     0,             -d2,          0,
↦ x2*d3+1/2*d1,1/2*d2,        -1/2*d3,      1/2
```

7.5.23.2 purityFiltration

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: purityFiltration(S), S matrix with entries of an Auslander regular ring D

Return: a list T of two lists, purity filtration of the module M=D^q/D^p(S^t)

Purpose: the first list T[1] gives a filtration {M_i} of M,
where the i-th entry of T[1] gives the representation matrix of M_(i-1).
the second list T[2] gives representations of the factor Modules,
i.e. T[2][i] gives the repr. matrix for M_(i-1)/M_i

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x1,x2,d1,d2),dp;
```

```
def S=Weyl();
setring S;
int i;
matrix R[3][3]=0,d2-d1,d2-d1,d2,-d1,-d1-d2,d1,-d1,-2*d1;
print(R);
↦ 0, -d1+d2,-d1+d2,
↦ d2,-d1,   -d1-d2,
↦ d1,-d1,   -2*d1
list T=purityFiltration(transpose(R));
// the purity filtration of coker(M)
print(T[1][1]);
↦ 0, -d1+d2,-d1+d2,
↦ d2,-d1,   -d1-d2,
↦ d1,-d1,   -2*d1
print(T[1][2]);
↦ d2,   d2,
↦ d1-d2,0,
↦ d2,   d1
print(T[1][3]);
↦ 1,0,
↦ 0,d2,
↦ 0,d1
// factor modules of the filtration
print(T[2][1]);
↦ 0, 1,1,
↦ -1,0,1
print(T[2][2]);
↦ 1,    1,
↦ d1-d2,0
print(T[2][3]);
↦ 1,0,
↦ 0,d2,
↦ 0,d1
```

7.5.23.3 purityTriang

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: purityTriang(S), S matrix with entries of an Auslander regular ring D

Return: a matrix T

Purpose: compute a triangular block matrix T, such that M=D^p/D^q(S^t) is isomorphic to M'=D^p'/D^q(T^t)

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x1,x2,d1,d2),dp;
def S=Weyl();
setring S;
int i;
matrix R[3][3]=0,d2-d1,d2-d1,d2,-d1,-d1-d2,d1,-d1,-2*d1;
print(R);
↦ 0, -d1+d2,-d1+d2,
↦ d2,-d1,   -d1-d2,
↦ d1,-d1,   -2*d1
matrix T=purityTriang(transpose(R));
// a triangular blockmatrix representing the module coker(R)
print(T);
↦ 0, 1,1,-1, 0,  0, 0,
↦ -1,0,1,0,  -1, 0, 0,
↦ 0, 0,0,-d1,-d2,-1,0,
↦ 0, 0,0,-1, -1, 0, -1,
↦ 0, 0,0,0,  0,  1, -d2,
↦ 0, 0,0,0,  0,  1, 0,
↦ 0, 0,0,0,  0,  0, d1
```

7.5.23.4 gradeNumber

Procedure from library `puritydfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: gradeNumber(R), R matrix, representing M=D^p/D^q(R^t) over a ring D

Return: int, grade number of M

Purpose: computes the grade number of M, i.e. the first i, with ext^i(M,D) !=0
returns -1 if M=0

Example:

```
LIB "purityfiltration.lib";
// trivial example
ring D=0,(x,y,z),dp;
matrix R[2][1]=1,x;
gradeNumber(R);
↦ 0
// R has left inverse, so M=D/D^2R=0
gradeNumber(transpose(R));
↦ -1
print(ncExt_R(0,R));
↦ 0
// so, ext^0(coker(R),D) =! 0)
//
// a little bit more complex
matrix R1[3][1]=x,-y,z;
gradeNumber(transpose(R1));
↦ 3
print(ncExt_R(0,transpose(R1)));
↦ 1
print(ncExt_R(1,transpose(R1)));
↦ 1
print(ncExt_R(2,transpose(R1)));
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
// ext^i are zero for i=0,1,2
matrix ext3=ncExt_R(3,transpose(R1));
print(ext3);
↦ z,y,x
// not zero
is_zero(ext3);
```

```
↦ 0
```

7.5.23.5 showgrades

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: showgrades(T), T list, which includes representation matrices of modules

Return: list, gradenumbers of the entries in T

Purpose: computes a list L with L[i]=gradenumber(M), M=D^p/D^qT[i]

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
list T=purityFiltration(transpose(R))[2];
showgrades(T);
↦ [1]:
↦    0
↦ [2]:
↦    -1
↦ [3]:
↦    2
↦ [4]:
↦    3
// T[i] are i-1 pure (i=1,3,4) or zero (i=2)
```

7.5.23.6 allExtOfLeft

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: allExtOfLeft(M),

Return: list, entries are ext-modules

Assume: M presents a left module of finite left projective dimension n

Purpose: For a left module presented by M over the basering D,
compute a list T, whose entry T[i+1] is a matrix, presenting the right module Ext^i_D(M,D) for i=0..n

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
// coker(R) consider the left module M=D^6/D^4R
list T=allExtOfLeft(transpose(R));
print(T[1]);
↦ 0
print(T[2]);
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
print(T[3]);
↦ 0,0, z,4y-z,4x,
↦ 0,-2,1,0,   1,
↦ 1,0, 0,0,   0,
↦ 0,1, 0,0,   0
print(T[4]);
↦ z,y,x
// right modules coker(T[i].)!!
```

7.5.23.7 allExtOfRight

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: allExtOfRight(R), R matrix representing the right Module M=D^q/RD^p over a ring D
M module with finite right projective dimension n

Return: list, entries are ext-modules

Purpose: computes a list T, which entries are representations of the left modules ext^i(M,D)
T[i] gives the repr. matrix of ext^(i-1)(M,D), i=1,..,n+1

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
// coker(R) considered as right module
projectiveDimension(R,1)[2];
↦ 1
list T=allExtOfRight(R);
print(T[1]);
↦ 4x,
↦ -4y,
↦ -z,
↦ z
print(T[2]);
↦ 1,0,0,   0,0,   0,
↦ 0,0,4y-z,0,4x-z,0,
↦ 0,z,0,   y,0,   x
// left modules coker(.T[i])!!
```

7.5.23.8 doubleExt

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: doubleExt(R,i), R matrix representing the left Module M=D^p/D^q(R^t) over a ring D
int i, less or equal the left projective dimension of M

Return: matrix P, representing the double ext module

Purpose: computes a matrix P, which represents the left module ext^i(ext^i(M,D))

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[7][3]=
0 ,0,1,
1 ,-4*x+z,-z,
-1,8*x-2*z,z,
1 ,0  ,0,
0 ,x-y,0,
0 ,x-y,y,
0 ,0  ,x;
// coker(R) is 2-pure, so all doubleExt are zero
print(doubleExt(transpose(R),0));
↦ 1
print(doubleExt(transpose(R),1));
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
print(doubleExt(transpose(R),3));
↦ 1
// except of the second
print(doubleExt(transpose(R),2));
↦ 4y-z,4x-z
```

7.5.23.9 allDoubleExt

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: allDoubleExt(R), R matrix representing the left Module M=D^p/D^q(R^t) over a ring D

Return: list T, double indexed, which include all double-ext modules

Purpose: computes all double ext-modules
T[i][j] gives a representation matrix of ext^(j-1)(ext(i-1)(M,D))

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x1,x2,x3,d1,d2,d3),dp;
def S=Weyl();
setring S;
matrix R[6][4]=
0,-2*d1,d3-2*d2-d1,-1,
0,d3-2*d1,2*d2-3*d1,1,
d3,-6*d1,-2*d2-5*d1,-1,
0,d2-d1,d2-d1,0,
d2,-d1,-d2-d1,0,
d1,-d1,-2*d1,0;
list T=allDoubleExt(transpose(R));
// left projective dimension of M=coker(R) is 3
// ext^i(ext^0(M,D)), i=0,1,2,3
print(T[1][1]);
↦ 0,
↦ d1,
↦ d3,
↦ -d2
print(T[1][2]);
↦ d3,d3,d2,d1
print(T[1][3]);
↦ 1
print(T[1][4]);
↦ 1
// ext^i(ext^1(M,D)), i=0,1,2,3
print(T[2][1]);
↦ 1
print(T[2][2]);
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
print(T[2][3]);
↦ 0,0,0,4*d2-d3,4*d1-d3,
↦ 1,0,0,0,      0,
↦ 0,1,0,0,      0,
↦ 0,0,1,0,      0
print(T[2][4]);
↦ d3,d2,d1
// ext^i(ext^2(M,D)), i=0,1,2,3  (all zero)
print(T[3][1]);
↦ 1
print(T[3][2]);
↦ 1
print(T[3][3]);
↦ 1
print(T[3][4]);
↦ 1
// ext^i(ext^3(M,D)), i=0,1,2,3  (all zero)
print(T[4][1]);
↦ 1
print(T[4][2]);
↦ 1
print(T[4][3]);
↦ 1
print(T[4][4]);
↦ 1
```

7.5.23.10 is_pure

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: is_pure(R), R representing the module M=D^p/D^q(R^t)

Return: int, 0 or 1

Purpose: checks pureness of M.
returns 1, if M is pure, or 0, if it's not
remark: if M is zero, is_pure returns 1

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[3][2]=y,-z,x,0,0,x;
list T=purityFiltration(transpose(R));
print(transpose(std(transpose(T[2][2]))));
↦ y,-z,
↦ x,0,
↦ 0,x
// so the purity filtration of coker(R) is trivial,
// i.e. coker(R) is already pure
is_pure(transpose(R));
↦ 1
// we can also have non-pure modules:
matrix R2[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
is_pure(transpose(R2));
↦ 0
```

7.5.23.11 purelist

Procedure from library `purityfiltration.lib` (see Section 7.5.23 [purityfiltration_lib], page 600).

Usage: purelist(T), T list, in which the i-th entry R=T[i] represents M=D^p/D^q(R^t)

Return: list M, entries of M are 0 or 1

Purpose: if T[i] is pure, M[i] is 1, else M[i] is 0

Example:

```
LIB "purityfiltration.lib";
ring D = 0,(x,y,z),dp;
matrix R[6][4]=
0,-2*x,z-2*y-x,-1,
0,z-2*x,2*y-3*x,1,
z,-6*x,-2*y-5*x,-1,
0,y-x,y-x,0,
y,-x,-y-x,0,
x,-x,-2*x,0;
is_pure(transpose(R));
```

```
↦ 0
// R is not pure, so we do the purity filtration
list T=purityFiltration(transpose(R));
// all Elements of T[2] are either zero or pure
purelist(T[2]);
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    1
```

7.5.24 qmatrix_lib

Library: qmatrix.lib

Purpose: Quantum matrices, quantum minors and symmetric groups

Authors: Lobillo, F.J., jlobillo@ugr.es
Rabelo, C., crabelo@ugr.es

Support: 'Metodos algebraicos y efectivos en grupos cuanticos', BFM2001-3141, MCYT, Jose Gomez-Torrecillas (Main researcher).

Procedures:

7.5.24.1 quantMat

Procedure from library `qmatrix.lib` (see Section 7.5.24 [qmatrix_lib], page 609).

Usage: quantMat(n [, p]); n integer (n>1), p an optional integer

Return: ring (of quantum matrices). If p is specified, the quantum parameter q
will be specialized at the p-th root of unity

Purpose: compute the quantum matrix ring of order n

Note: activate this ring with the "setring" command.
The usual representation of the variables in this quantum
algebra is not used because double indexes are not allowed
in the variables. Instead the variables are listed by reading
the rows of the usual matrix representation, that is, there
will be n*n variables (one for each entry an n*N generic matrix),
listed row-wise

Example:

```
LIB "qmatrix.lib";
def r = quantMat(2); // generate O_q(M_2) at q generic
setring r;   r;
↦ // coefficients: QQ(q)
↦ // number of vars : 4
↦ //        block   1 : ordering Dp
↦ //                  : names    y(1) y(2) y(3) y(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
```

```
↦ //    y(2)y(1)=1/(q)*y(1)*y(2)
↦ //    y(3)y(1)=1/(q)*y(1)*y(3)
↦ //    y(4)y(1)=y(1)*y(4)+(-q^2+1)/(q)*y(2)*y(3)
↦ //    y(4)y(2)=1/(q)*y(2)*y(4)
↦ //    y(4)y(3)=1/(q)*y(3)*y(4)
kill r;
def r = quantMat(2,5); // generate O_q(M_2) at q^5=1
setring r;   r;
↦ // coefficients: QQ[q]/(q^4+q^3+q^2+q+1)
↦ // number of vars : 4
↦ //        block   1 : ordering Dp
↦ //                  : names    y(1) y(2) y(3) y(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    y(2)y(1)=(-q^3-q^2-q-1)*y(1)*y(2)
↦ //    y(3)y(1)=(-q^3-q^2-q-1)*y(1)*y(3)
↦ //    y(4)y(1)=y(1)*y(4)+(-q^3-q^2-2*q-1)*y(2)*y(3)
↦ //    y(4)y(2)=(-q^3-q^2-q-1)*y(2)*y(4)
↦ //    y(4)y(3)=(-q^3-q^2-q-1)*y(3)*y(4)
```

See also: Section 7.5.24.2 [qminor], page 610.

7.5.24.2 qminor

Procedure from library `qmatrix.lib` (see Section 7.5.24 [qmatrix_lib], page 609).

Usage: qminor(I,J,n); I,J intvec, n int

Return: poly, the quantum minor of a generic n*n quantum matrix

Assume: I is the ordered list of the rows to consider in the minor,
J is the ordered list of the columns to consider in the minor,
I and J must have the same number of elements,
n is the order of the quantum matrix algebra you are working with (quantMat(n)).
The base ring should be constructed using `quantMat`.

Example:

```
LIB "qmatrix.lib";
def r = quantMat(3); // let r be a quantum matrix of order 3
setring r;
intvec u = 1,2;
intvec v = 2,3;
intvec w = 1,2,3;
qminor(w,w,3);
↦ y(1)*y(5)*y(9)+(-q)*y(1)*y(6)*y(8)+(-q)*y(2)*y(4)*y(9)+(q^2)*y(2)*y(6)*y(\
   7)+(q^2)*y(3)*y(4)*y(8)+(-q^3)*y(3)*y(5)*y(7)
qminor(u,v,3);
↦ y(2)*y(6)+(-q)*y(3)*y(5)
qminor(v,u,3);
↦ y(4)*y(8)+(-q)*y(5)*y(7)
qminor(u,u,3);
↦ y(1)*y(5)+(-q)*y(2)*y(4)
```

See also: Section 7.5.24.1 [quantMat], page 609.

7.5.24.3 SymGroup

Procedure from library `qmatrix.lib` (see Section 7.5.24 [qmatrix_lib], page 609).

Usage: SymGroup(n); n an integer (positive)

Return: intmat

Purpose: represent the symmetric group S(n) via integer vectors (permutations)

Note: each row of the output integer matrix is an element of S(n)

Example:

```
LIB "qmatrix.lib";
//  "S(3)={(1,2,3),(1,3,2),(3,1,2),(2,1,3),(2,3,1),(3,2,1)}";
SymGroup(3);
↦ 1,2,3,
↦ 1,3,2,
↦ 3,1,2,
↦ 2,1,3,
↦ 2,3,1,
↦ 3,2,1
```

See also: Section 7.5.24.5 [LengthSym], page 611; Section 7.5.24.4 [LengthSymElement], page 611.

7.5.24.4 LengthSymElement

Procedure from library `qmatrix.lib` (see Section 7.5.24 [qmatrix_lib], page 609).

Usage: LengthSymElement(v); v intvec

Return: int

Purpose: determine the length of the permutation given by v in some S(n)

Assume: v represents an element of S(n); otherwise the output may have no sense

Example:

```
LIB "qmatrix.lib";
intvec v=1,3,4,2,8,9,6,5,7,10;
LengthSymElement(v);
↦ 9
```

See also: Section 7.5.24.5 [LengthSym], page 611; Section 7.5.24.3 [SymGroup], page 611.

7.5.24.5 LengthSym

Procedure from library `qmatrix.lib` (see Section 7.5.24 [qmatrix_lib], page 609).

Usage: LengthSym(M); M an intmat

Return: intvec

Purpose: determine a vector, where the i-th element is the length of the permutation of S(n) given by the i-th row of M

Assume: M represents a subset of S(n) (each row must be an element of S(n)); otherwise, the output may have no sense

Example:

```
LIB "qmatrix.lib";
def M = SymGroup(3); M;
↦ 1,2,3,
↦ 1,3,2,
↦ 3,1,2,
↦ 2,1,3,
↦ 2,3,1,
↦ 3,2,1
LengthSym(M);
↦ 0,1,2,1,2,3
```

See also: Section 7.5.24.4 [LengthSymElement], page 611; Section 7.5.24.3 [SymGroup], page 611.

7.5.25 ratgb_lib

Status: experimental

Library: ratgb.lib

Purpose: Groebner bases in Ore localizations of noncommutative G-algebras

Author: Viktor Levandovskyy, levandov@risc.uni-linz.ac.at

Overview: Theory: Let A be an operator algebra with `R = K[x1,.,xN]` as subring. The operators are usually denoted by `d1,..,dM`.

Assume, that A is a `G`-algebra, then the set `S=R-0` is multiplicatively closed Ore set in A. That is, for any s in S and a in A, there exist t in S and b in A, such that `sa=bt`. In other words, one can transform any left fraction into a right fraction. The algebra `A_S` is called an Ore localization of A with respect to S.

This library provides Groebner basis procedure for A_S, performing polynomial (that is fraction-free) computations only. Note, that there is ongoing development of the subsystem called Singular:Locapal, which will provide yet another approach to Groebner bases over such Ore localizations.

Assumptions: in order to treat such localizations constructively, some care need to be taken. We will assume that the variables `x1,...,xN` from above (which will become invertible in the localization) come as the first block among the variables of the basering. Moreover, the ordering on the basering must be an antiblock ordering, that is its matrix form has the left upper `NxN` block zero. Here is a recipe to create such an ordering easily: use 'a(w)' definitions of the ordering N times with intvecs `w_i` of the following form: `w_i` has first N components zero. The rest entries need to be positive and such, that `w1,..,wN` are linearly independent (see an example below).

Guide: with this library, it is possible
- to compute a Groebner basis of an ideal or a submodule in the 'rational' Ore localization D = A_S
- to compute a dimension of associated graded submodule (called D-dimension) - to compute a vector space dimension over Quot(R) of a submodule of D-dimension 0 (so called D-finite submodule)
- to compute a basis over Quot(R) of a D-finite submodule

Procedures: See also: Section D.11.3 [jacobson_lib], page 1977; Section 7.5.21 [olga_lib], page 581.

7.5.25.1 ratstd

Procedure from library `ratgb.lib` (see Section 7.5.25 [ratgb_lib], page 612).

Usage: ratstd(I, n [,eng]); I an ideal/module, n an integer, eng an optional integer

Return: ring

Purpose: compute the Groebner basis of I in the Ore localization of the basering with respect to the subalgebra, generated by first n variables

Assume: the variables of basering are organized in two blocks and - the first block of length `n` contains the elements with respect to which one localizes, - the basering is equipped with anti-block ordering, giving block dominance for the variables in the second block

Note: the output ring C is commutative. The ideal `rGBid` in C represents the rational form of the output ideal `pGBid` in the basering. - During the computation, the D-dimension of I, `Ratgb::Ddim` and the corresponding dimension as K(x)-vector space of I (`Ratgb::KXdim`, if `Ratgb::Ddim`=0) are computed and exported. - Setting optional integer eng to 1, std is taken as Groebner engine; default is slimgb.

Display: In order to see the steps of the computation, set printlevel to >=2

Example:

```
LIB "ratgb.lib";
ring r = (0,c),(x,y,Dx,Dy),(a(0,0,1,1),a(0,0,1,0),dp);
// this ordering is an antiblock ordering, as it must be
def S = Weyl(); setring S;
// the ideal I below annihilates parametric Appel F4 function
// where we set parameters to a=-2, b=-1 and d=0
ideal I =
x*Dx*(x*Dx+c-1) - x*(x*Dx+y*Dy-2)*(x*Dx+y*Dy-1),
y*Dy*(y*Dy-1) - y*(x*Dx+y*Dy-2)*(x*Dx+y*Dy-1);
int is = 2; // hence 1st and 2nd variables, that is x and y
// will become invertible in the localization
def A = ratstd(I,2); // main call
pGBid; // polynomial form of the basis in the localized ring
↦ pGBid[1]=2*x*y*Dx*Dy+x*y*Dy^2+y^2*Dy^2-y*Dy^2+(-c-2)*x*Dx-2*y*Dy+2
↦ pGBid[2]=x*Dx^2-y*Dy^2+(c)*Dx
↦ pGBid[3]=2*x^2*y*Dy^3-4*x*y^2*Dy^3+2*y^3*Dy^3-4*x*y*Dy^3-4*y^2*Dy^3+2*y*D\
   y^3+(-2*c)*x^2*Dx*Dy+(2*c)*x*Dx*Dy+2*x^2*Dy^2+(c-2)*x*y*Dy^2+(-3*c)*y^2*D\
   y^2-4*x*Dy^2+(3*c-2)*y*Dy^2+2*Dy^2+(-c^2+2*c)*x*Dx+(6*c)*y*Dy+(-6*c)
setring A; // A is a commutative ring used for presentation
rGBid; // "rational" or "localized" form of the basis
↦ rGBid[1]=(2*x*y)*Dx*Dy+(x*y+y^2-y)*Dy^2+(-c*x-2*x)*Dx+(-2*y)*Dy+2
↦ rGBid[2]=(x)*Dx^2+(-y)*Dy^2+(c)*Dx
↦ rGBid[3]=(2*x^2*y-4*x*y^2-4*x*y+2*y^3-4*y^2+2*y)*Dy^3+(-2*c*x^2+2*c*x)*Dx\
   *Dy+(c*x*y-3*c*y^2+3*c*y+2*x^2-2*x*y-4*x-2*y+2)*Dy^2+(-c^2*x+2*c*x)*Dx+(6\
   *c*y)*Dy+(-6*c)
Ratgb::Ddim; // the Krull-like dimension of A/I
↦ 0
Ratgb::KXdim; // the dimension of A/I as a left K(x,y)-vector space
↦ 4
//--- Now, let us compute a K(x,y) basis explicitly
print(matrix(kbase(rGBid)));
↦ // ** rGBid is no standard basis
↦ Dy^2,Dy,Dx,1
```

7.6 Graded commutative algebras (SCA)

This section describes basic mathematical notions, definition, and a little bit the implementation of the experimental non-commutative kernel extension Sca of Singular which improves performance of many algorithms in graded commutative algebras.

In order to improve performance of Singular in specific non-commutative algebras one can extend the internal implementation for them in a virtual-method-overloading-like manner. At the moment graded commutative algebras (SCA) and in particular exterior algebras are implemented this way.

Note that graded commutative algebras require no special user actions apart from defining an appropriate non-commutative GR-algebra in Singular. Upon doing that, the supper-commutative structure will be automatically detected and special multiplication will be used. Moreover, in most Sca-aware (e.g. `std`) algorithms special internal improvements will be used (otherwise standard generic non-commutative implementations will be used).

All considered algebras are assumed to be associative K -algebras for some ground field K .

Definition

Polynomial graded commutative algebras are factors of tensor products of commutative algebras with an exterior algebra over a ground field K .

These algebras can be naturally endowed with a $Z/2Z$ -grading, where anti-commutative algebra generators have degree 1 and commutative algebra generators (and naturally scalars) have degree 0 . In this particular case they may be considered as super-commutative algebras.

GR-algebra representation

A graded commutative algebra with n commutative and m anti-commutative algebra generators can be represented as factors of the following GR-algebra by some two-sided ideal:

$$K\left\langle x_1,\ldots,x_n; y_1,\ldots,y_m \mid y_j * y_i = -y_i y_j, i<j \right\rangle / \left\langle y_1^2,\ldots,y_m^2 \right\rangle .$$

Distinctive features

Graded commutative algebras are Noetherian.

Graded commutative algebras have zero divisors if and only if $m > 0 : \; y_i * y_i = 0$.

Unlike other non-commutative algebras one may use any monomial ordering where only the non-commutative variables are required to be global. In particular, commutative variables are allowed to be local. This means that one can work in tensor products of any commutative ring with an exterior algebra.

Example of defining graded commutative algebras in Singular:SCA and computing with them

Given a commutative polynomial ring r , super-commutative structure on it can be introduced as follows:

```
LIB "nctools.lib";
ring r = 0,(a, b, x,y,z, Q, W),(lp(2), dp(3), Dp(2));
// Let us make variables x = var(3), ..., z = var(5) to be anti-commutative
// and add additionally a quotient ideal:
def S = superCommutative(3, 5, ideal(a*W + b*Q*x + z) ); setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 7
↦ //        block   1 : ordering lp
↦ //                  : names    a b
↦ //        block   2 : ordering dp
↦ //                  : names    x y z
↦ //        block   3 : ordering Dp
↦ //                  : names    Q W
↦ //        block   4 : ordering C
↦ // noncommutative relations:
↦ //    yx=-xy
↦ //    zx=-xz
↦ //    zy=-yz
↦ // quotient ring from ideal
↦ _[1]=xz
↦ _[2]=bxyQ-yz
↦ _[3]=aW+bxQ+z
↦ _[4]=z2
↦ _[5]=y2
↦ _[6]=x2
ideal I = a*x*y + z*Q + b, y*Q + a; I;
↦ I[1]=axy+b+zQ
↦ I[2]=a+yQ
std(I); // Groebner basis is used here since > is global
↦ _[1]=yQW-z
↦ _[2]=yz
↦ _[3]=b+zQ
↦ _[4]=a+yQ
kill r, S;
// Let’s do the same but this time with some local commutative variables:
ring r = 0,(a, b, x,y,z, Q, W),(dp(1), ds(1), lp(3), ds(2));
def S = superCommutative(3, 5, ideal(a*W + b*Q*x + z) ); setring S; S;
↦ // coefficients: QQ
↦ // number of vars : 7
↦ //        block   1 : ordering dp
↦ //                  : names    a
↦ //        block   2 : ordering ds
↦ //                  : names    b
↦ //        block   3 : ordering lp
↦ //                  : names    x y z
↦ //        block   4 : ordering ds
↦ //                  : names    Q W
↦ //        block   5 : ordering C
↦ // noncommutative relations:
↦ //    yx=-xy
↦ //    zx=-xz
↦ //    zy=-yz
↦ // quotient ring from ideal
↦ _[1]=xz
↦ _[2]=yz-bxyQ
↦ _[3]=aW+z+bxQ
↦ _[4]=x2
↦ _[5]=y2
↦ _[6]=z2
ideal I = a*x*y + z*Q + b, y*Q + a; I;
↦ I[1]=axy+zQ+b
↦ I[2]=a+yQ
std(I);
↦ _[1]=yQW-z-bxQ
↦ _[2]=zQ+b
↦ _[3]=bx
↦ _[4]=by
↦ _[5]=bz
↦ _[6]=b2
↦ _[7]=a+yQ
```

See example of Section 7.5.20.9 [superCommutative], page 568 from the library `nctools.lib`.

Reference: Ph.D thesis by Oleksandr Motsak (2010), `https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-26479`.

7.7 LETTERPLACE

This section describes mathematical notions and definitions used in the Letterplace subsystem of Singular.

All algebras are assumed to be associative R -algebras for R being a field K

or a ring Z .

What is and what does Letterplace?

What is Letterplace? It is a subsystem of Singular, providing the manipulations and computations within free associative algebras over rings R

$< x_1 , \ldots , x_n >$, where the coefficient domain R is either a ring Z or a field, supported by Singular.

Letterplace can perform computations also in the factor-algebras of the above (via data type `qring`) by two-sided ideals.

Free algebras are internally represented in Singular as so-called Letterplace rings.

Each such ring is constructed from a commutative ring R [x_1 ,. . . , x_n] and a **degree (length) bound** d .

This encodes a sub- K -vector space (also called a filtered part) of K

$< x_1 , \ldots , x_n >$, spanned by all monomials of **length** at most d . Analogously for free R -subbimodules of a free R -bimodule of a fixed rank.

Within such a construction we offer the computations of Groebner (also known as Groebner-Shirshov) bases, normal forms, syzygies and many more.

We address both two-sided ideals and subbimodules of the free bimodule of the fixed rank.

A variety of monomial and module orderings is supported, including **elimination** orderings for both variables and bimodule components. A monomial ordering has to be a well-ordering.

Letterplace works with every field, supported by Singular, and with the coefficient ring Z .

Note, that the elements of the coefficient field (or a ring) mutually commute with all variables.

7.7.1 Examples of use of Letterplace

First, define a commutative ring $K[X]$ in Singular, equipped with a monomial well-ordering and call it, say, `r`.

Then, decide what should be the degree (length) bound d , that is how long may the words (monomials in the free algebra) become and run the procedure `freeAlgebra(r, d)`.

In the case you wish to work with subbimodules of the free bimodule of rank k , use freeAlgebra(r, d, k) instead of the previous.

The freeAlgebra. procedure creates free algebra $K < X >$ resp. the free bimodule of rank k over $K < X >$ subject to a monomial (module) ordering, corresponding to the one in the original commutative ring $K[X]$, see Section 7.9.2 [Monomial orderings on free algebras], page 636.

Polynomial (vector) arithmetics in this K -algebra is the usual one: +,-,*,^ while of course, x*y and y*x are different monomials while x*7=7*x.

Let us define an ideal I as a list of polynomials in the free algebra and run, for example, twostd (see Section 7.8.14 [twostd (letterplace)], page 634). The answer is a two-sided Groebner basis J of the two-sided ideal I

up to the length bound d.

Then, we want to compute the following: 1. The two-sided normal form of xyzy with respect to J using the function reduce (see Section 7.8.9 [reduce (letterplace)], page 631). 2. By introducing a factor algebra $K < x, y, z > /J$ of type qring, and demonstrate the functions reduce and rightstd (for right Groebner bases) over the factor algebra. 3. By creating the free R-bimodule of rank 8, we demonstrate how embeddins works with imap and also, how to express a subbimodule (or a single element) in terms of bimodule generators with lift. In other words, we compute and compare presentations of a polynomials with respect to the original generating set of ideal and with respect to a Groebner basis. 4. In the same free R-bimodule we will compute the module of bisyzygies of J and do some syzygy tests. 5. We demonstrate the bimodule membership problem: a boolean answer via NF and the certified version (with a Groebner presentation) via lift. 6. We show how elimination of module components works for bimodules.

We illustrate the approach with the following example:

```
//******* Part 1 *******//
LIB "freegb.lib";
ring r = 0,(x,y,z),dp; // the ordering on the free algebra will be degree right lex
ring R = freeAlgebra(r, 5);  // 5 the is degree (length) bound;
ideal I = x*y + y*z, x*x + x*y - z; // define an ideal via the set of polynomials
ideal J = twostd(I);
J; // as we see, with respect to the current ordering this Groebner basis
↦ J[1]=x*y+y*z
↦ J[2]=x*x-y*z-z
↦ J[3]=y*z*y-y*z*z+z*y
↦ J[4]=y*z*x+y*z*z+z*x-x*z
↦ J[5]=y*z*z*y-y*z*z*z-x*z*y
↦ J[6]=y*z*z*x+y*z*z*z-x*z*x+y*z*z+z*z
↦ J[7]=y*z*z*z*y-y*z*z*z*z+y*z*z*z+x*z*y+z*z*y
↦ J[8]=y*z*z*z*x+y*z*z*z*z+x*z*x+z*z*x-x*z*z-y*z*z-z*z
// tends to be infinite. Increasing the bound and recomputing helps to check it.
poly p = reduce(x*y*z*y,J);
p; // since p!=0, x*y*z*y is not contained in J up to length 5
↦ -y*z*z*z-x*z*y
// however this does not imply a definite answer on whether p is in J
poly q = x*(y+1)*z*y-x*y*z^2;
reduce(q, J); // 0, thus q is in J
↦ 0
//******* Part 2 *******//
qring Q = J; // J is a Groebner basis, computed above
poly p = reduce(x*x, twostd(0)); // the canonical representative of x*x in Q
p;
```

```
↦ y*z+z
rightstd(ideal(p)); // right Groebner basis of the right ideal, generated by p in Q
↦ _[1]=z*z
↦ _[2]=y*z+z
↦ _[3]=x*z
//******* Part 3 *******//
setring r;
ring R8 = freeAlgebra(r, 5, 8);  // 5 the is length bound; 8 is the rank of the free
ideal J = imap(R, J); // we map J identically from R (of rank 1)
J = twostd(J);
poly q = imap(R, q);
NF(q, J); // NF is an alias to reduce, we have rechecked that q is in J
↦ 0
matrix L = lift(J, q); // creates the presentation for q in terms of J
// since J is a Groebner basis, this is a Groebner presentation of q
print(transpose(matrix(L))); // J has 8 generators and these are the needed coefficie
↦ ncgen(1)*z*y-ncgen(1)*z*z,0,0,0,-ncgen(5),0,0,0
// here, the generators of the free bimodule are ncgen(1)*gen(1), ... , ncgen(8)*gen
// the output means, that substituting ncgen(i) by the i-th generator of J, we get q
J[1]*z*y - J[1]*z*z  - J[5] - q; // 0, so this is the seeked expression of q
↦ 0
testLift(J,L); // recovers q from the lift matrix
↦ _[1]=x*y*z*y-x*y*z*z+x*z*y
// Let us compare now this nice Groebner presentation with the one
// obtained from the original set of generators
ideal I = imap(R,I); // note: I is not a Groebner basis of itself
matrix M = lift(I, q); // creates the presentation for q in terms of I
M; // presentation is longer and more complicated than the one in L
↦ M[1,1]=-ncgen(1)*x*y+x*ncgen(1)*y-ncgen(1)*y*z-x*ncgen(1)*z+y*z*ncgen(1)+\
   z*ncgen(1)
↦ M[2,1]=ncgen(2)*x*y-x*ncgen(2)*y+ncgen(2)*y*z
testLift(I,M); // a routine test to ensure that indeed we recover q
↦ _[1]=x*y*z*y-x*y*z*z+x*z*y
//******* Part 4 *******//
// Let us compute the module of bisyzygies of J and analyze it
module S = syz(J); size(S); // 18
↦ 18
S[6]; // consider, for example, this element
↦ ncgen(1)*z*y*gen(1)-ncgen(1)*z*z*gen(1)+y*z*ncgen(1)*gen(1)-ncgen(4)*y*ge\
   n(4)-ncgen(3)*z*gen(3)+z*ncgen(1)*gen(1)-x*ncgen(3)*gen(3)
// plugging the i-th generator of J instead of ncgen(i), we obtain a bisyzygy:
J[1]*z*y - J[1]*z*z - x*J[3] - J[5]; //0
↦ 0
module S2 = S[6..8]; // pick just three generators
testSyz(J,S2); // tests the bisyzygy property for the generators
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
//******* Part 5 *******//
option(redSB); option(redTail); // to compute minimal and tail-reduced bases
module GS = twostd(S); size(GS); // 30
↦ 30
// let us construct a vector, belonging to GS:
```

```
vector v = GS[11]*y - x*GS[7] + z*GS[3]*z;
print(v);
↦ [-x*ncgen(1)*x*y-x*ncgen(1)*y*z+x*x*y*ncgen(1)-x*z*ncgen(1),y*z*z*ncgen(2\
   )*y+z*y*z*ncgen(2)*z-x*z*ncgen(2)*y+z*z*ncgen(2)*z,z*ncgen(3)*z*z+x*ncgen\
   (3)*z,-z*ncgen(4)*x*z+x*ncgen(4)*y,ncgen(5)*z*y-x*ncgen(5),-ncgen(6)*x*y+\
   z*ncgen(6)*z+ncgen(6)*y,0,ncgen(8)*y]
NF(v, GS); // 0, by the construction
↦ 0
// now we wish to compute the expression of v via GS
ring r3 = 0,(x,y,z),(c,dp);
ring R30 = freeAlgebra(r3,5,30);
module GS = imap(R8,GS);
vector v = imap(R8,v);
matrix L = lift(GS,v); // via printing we see only three components involved:
L[3,1]; // =z*ncgen(3)*z, as well as
↦ z*ncgen(3)*z
L[7,1]; // =-x*ncgen(7) and
↦ -x*ncgen(7)
L[11,1]; // =ncgen(11)*y
↦ ncgen(11)*y
//******* Part 6 *******//
// Notice, that the module ordering is (c,dp): it is a position-over-term ordering
// which eliminates module components in an descending way.
GS = GS[1..5]; // consider just first five syzygies,  for a smaller example
GS = twostd(GS); // a nice finite Groebner basis
print(matrix(GS)); // shows the structure
↦ 0,       0,       0,     0,     0,     _[1,6],   _[1,7],   _[1,8],
↦ 0,       _[2,2],  _[2,3],_[2,4],_[2,5],_[2,6],   _[2,7],   0,
↦ _[3,1],  _[3,2],  _[3,3],_[3,4],_[3,5],-ncgen(3),0,        _[3,8],
↦ 0,       _[4,2],  _[4,3],_[4,4],_[4,5],-ncgen(4),-ncgen(4),_[4,8],
↦ ncgen(5),0,       0,     _[5,4],_[5,5],0,        0,        ncgen(5),
↦ 0,       ncgen(6),0,     _[6,4],_[6,5],0,        0,        0,
↦ ncgen(7),0,       0,     0,     _[7,5],0,        0,        0
// As we can see, intersections of the subbimodule GS with the free bimodules
// generated by all but first resp. all but first two bimodule generators
// are not empty and given by vectors having zero in the first resp.
// in the first two components.
```

See Section 7.8 [Functions (letterplace)], page 624 for the list of all available kernel functions.

There are various conversion routines in the library `freegb_lib` (see Section 7.10.4 [freegb_lib], page 665). Many algebras are predefined in the library `fpalgebras_lib` (see Section 7.10.2 [fpalgebras_lib], page 645). Important ring-theoretic properties can be established with the help of the library `fpaprops_lib` (see Section 7.10.3 [fpaprops_lib], page 659), while K-dimension and monomial bases and Hilbert data - with the help of the library `fpadim_lib` (see Section 7.10.1 [fpadim_lib], page 639). We work further on implementing more algorithms for non-commutative ideals and modules over free associative algebra.

7.7.2 Example of use of Letterplace over Z

Consider the following paradigmatic example:

```
LIB "freegb.lib";
ring r = integer,(x,y),Dp;
ring R = freeAlgebra(r,5); // length bound is 5
```

```
ideal I = 2*x, 3*y;
I = twostd(I);
print(matrix(I)); // pretty prints the generators
↦ 3*y,2*x,y*x,x*y
```

As we can see, over $Z < x, y >$ the ideal $< 2x, 3y >$ has a finite Groebner basis and indeed

$Z < x, y > / < 2x, 3y >=$

$Z < x, y > / < 2x, 3y, yx, xy >=$

$Z < x, y > / < 2x, 3y, yx - xy, xy >$

and the later is naturally isomorphic to

$Z[x, y]/ < 2x, 3y, xy >$ as a Z -algebra.

Now, we analyze the same ideal in the ring with one more variable z :

```
LIB "freegb.lib";
ring r = integer,(x,y,z),Dp;
ring R = freeAlgebra(r,5); // length bound is 5
ideal I = 2*x, 3*y;
I = twostd(I);
print(matrix(I)); // pretty prints the generators
↦ 3*y,2*x,y*x,x*y,y*z*x,x*z*y,y*z*z*x,x*z*z*y,y*z*z*z*x,x*z*z*z*y
```

Now we see, that this Groebner basis is potentially infinite and the following argument delivers a proof. Namely, yz^ix and

xz^iy are present in the ideal for all $i >= 0$. How can we do this? We wish to express $y * z^i * x$ and $x * z^i * y$ via the original generators by means of `lift`:

```
LIB "freegb.lib";
ring r = integer,(x,y,z),Dp;
ring R = freeAlgebra(r,5,2); // length bound is 5, rank of the free bimodule is 2
ideal I = 2*x, 3*y;
matrix T1 = lift(I, ideal(y*z*x,x*z*y));
print(T1);
↦ -y*z*ncgen(1),-ncgen(1)*z*y,
↦ ncgen(2)*z*x, x*z*ncgen(2)
-y*z*I[1] + I[2]*z*x; // gives y*z*x
↦ y*z*x
matrix T2 = lift(I, ideal(y*z^2*x,x*z^2*y));
print(T2);
↦ -y*z*z*ncgen(1),-ncgen(1)*z*z*y,
↦ ncgen(2)*z*z*x, x*z*z*ncgen(2)
-y*z^2*I[1] + I[2]*z^2*x; // gives y*z^2*x
↦ y*z*z*x
```

The columns of matrices, returned by `lift`, encode the presentation of new elements in terms of generators. From this we conjecture, that in particular

$-yz^i * (2x) + (3y) * z^ix = yz^ix$ holds for all $i >= 0$

and indeed, confirm it via a routine computation by hands.

Comparing computations over Q with computations over Z.

In the next example, we first compute over the field of rationals Q and a bit later compare the result with computations over the ring of integers Z .

```
LIB "freegb.lib"; // initialization of free algebras
ring r = 0,(z,y,x),Dp; // degree left lex ord on z>y>x
ring R = freeAlgebra(r,7); // length bound is 7
```

```
ideal I = y*x - 3*x*y - 3*z, z*x - 2*x*z +y, z*y-y*z-x;
option(redSB); option(redTail); // for minimal reduced GB
option(intStrategy); // avoid divisions by coefficients
ideal J = twostd(I); // compute a two-sided GB of I
J; // prints generators of J
↦ J[1]=4*x*y+3*z
↦ J[2]=3*x*z-y
↦ J[3]=4*y*x-3*z
↦ J[4]=2*y*y-3*x*x
↦ J[5]=2*y*z+x
↦ J[6]=3*z*x+y
↦ J[7]=2*z*y-x
↦ J[8]=3*z*z-2*x*x
↦ J[9]=4*x*x*x+x
LIB "fpadim.lib"; // load the library for K-dimensions
lpMonomialBasis(7,0,J); // all monomials of length up to 7 in Q<x,y,z>/J
↦ _[1]=1
↦ _[2]=z
↦ _[3]=y
↦ _[4]=x
↦ _[5]=x*x
```

As we see, we obtain a nice finite Groebner basis J. Moreover, from the form of its leading monomials, we conjecture that

$Q < x,y,z > /J$ is finite dimensional Q -vector space. We check it with lpMonomialBasis and obtain an affirmative answer.

Now, for doing similar computations over Z one needs to change only the initialization of the ring, the rest stays the same

```
LIB "freegb.lib"; // initialization of free algebras
ring r = integer,(z,y,x),Dp; // Z and deg left lex ord on z>y>x
ring R = freeAlgebra(r,7); // length bound is 7
ideal I = y*x - 3*x*y - 3*z, z*x - 2*x*z +y, z*y-y*z-x;
option(redSB); option(redTail); // for minimal reduced GB
option(intStrategy); // avoid divisions by coefficients
ideal J = twostd(I); // compute a two-sided GB of I
J; // prints generators of J
↦ J[1]=12*x*y+9*z
↦ J[2]=9*x*z-3*y
↦ J[3]=y*x-3*x*y-3*z
↦ J[4]=6*y*y-9*x*x
↦ J[5]=6*y*z+3*x
↦ J[6]=z*x-2*x*z+y
↦ J[7]=z*y-y*z-x
↦ J[8]=3*z*z+2*y*y-5*x*x
↦ J[9]=6*x*x*x-3*y*z
↦ J[10]=4*x*x*y+3*x*z
↦ J[11]=3*x*x*z+3*x*y+3*z
↦ J[12]=2*x*y*y+75*x*x*x+39*y*z+39*x
↦ J[13]=3*x*y*z-3*y*y+6*x*x
↦ J[14]=2*y*y*y+x*x*y+3*x*z
↦ J[15]=2*x*x*x*x+y*y-x*x
↦ J[16]=2*x*x*x*y+3*y*y*z+3*x*y+3*z
↦ J[17]=x*x*y*z+x*y*y-x*x*x
↦ J[18]=x*y*y*z-y*y*y+x*x*y
↦ J[19]=x*x*x*x*x+y*y*y*z+x*x*x
↦ J[20]=x*x*x*x*z+x*x*x*y+2*y*y*z+x*x*z+3*x*y+3*z
↦ J[21]=x*y*y*y*z-y*y*y*y+x*x*x*x-y*y+x*x
↦ J[22]=y*y*y*z*z-x*x*x*x*y
↦ J[23]=x*y*y*y*y*z-y*y*y*y*y+x*x*y*y*y
↦ J[24]=x*y*y*y*y*y*z-y*y*y*y*y*y+x*x*x*x*y*y+y*y*y*y+x*x*x*x+2*y*y-2*x*x
```

The output has plenty of elements in each degree (which is the same as length because of the degree ordering), what hints at potentially infinite Groebner basis.

Indeed, one can show that for every $i >= 2$ the ideal J contains an element with the leading monomial xy^iz .

7.7.3 Functionality and release notes of Letterplace

Over free associative algebras over fields or over a ring Z , one can perform many different computations with arbitrary two-sided ideals. It is possible to define a free bimodule of a fixed finite rank and also work with subbimodules of such. Groebner bases and related tools are thoroughly implemented, with respect to a variety of monomial module orderings.

The variables can be weighted by nonnegative weights, which are determined by the monomial ordering.

Restrictions/conventions of the Letterplace subsystem:

> Since free algebra is not Noetherian, one has to work with explicitly fixed degree (length) bound, up to which a partial Groebner basis will be computed. The initialization routine `freeAlgebra (letterplace)` constructs the ring with this bound. For increasing the length bound one needs to define another ring and to use `imap` for mapping the objects back and forth.
>
> All the computations happen up to the length bound, which is explicitly fixed during the definition of the current ring.
>
> The options `redSB, redTail` are effective for computations involving Groebner bases,
>
> The options `prot, mem` are effective for the whole Letterplace subsystem.
>
> For monomial orderings, which are not compatible with the length, the following error message might appear: `degree bound of Letterplace ring is 11, but at least 12 is needed for this multiplication` In such a situation, activating `option(redSB)`, `option(redTail)` and increasing the length (degree) bound might help. Though there are situations, where nothing leads to a finite computation simply while the nature of non-Noetherian rings is so.

Operations for polynomials in Letterplace rings are the usual ones: `+` (addition), `-` (subtraction), `*` (multiplication) and `^` (power).

The functions Section 7.3.2 [bracket], page 335, Section 5.1.88 [maxideal], page 219 and Section 5.1.151 [std], page 271 (an alias for Section 7.8.14 [twostd (letterplace)], page 634) also work within letterplace rings:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp; // the ordering will be degree right lex
ring R = freeAlgebra(r, 5);  // degree (length) bound is 5
// maxideal in a letterplace ring:
print(matrix(maxideal(2))); // all monomials of length 2
↦ x*x,y*x,z*x,x*y,y*y,z*y,x*z,y*z,z*z
// bracket in a letterplace ring:
bracket(x,y);
```

```
↦ -y*x+x*y
poly f = x*x + x*y - z;
bracket(f,x);
↦ x*y*x-x*x*y-z*x+x*z
bracket(f,x,2); // left-normed iterated bracket [f,[f,x]]
↦ -x*y*x*x*x+x*x*y*x*x+x*x*x*y*x+x*y*x*y*x-x*x*x*x*y-2*x*y*x*x*y+x*x*y*x*y+\
   z*x*x*x-x*z*x*x-z*x*y*x-x*x*z*x-x*y*z*x+2*z*x*x*y-x*z*x*y+x*x*x*z+2*x*y*x\
   *z-x*x*y*z+z*z*x-2*z*x*z+x*z*z
```

Further functionality is provided in the libraries for the Letterplace subsystem: see Section 7.10 [LETTERPLACE libraries], page 639 for details.

In the Section 7.10.4 [freegb_lib], page 665 one finds e.g. Letterplace initialization together with legacy, conversion and convenience tools.

The Section 7.10.1 [fpadim_lib], page 639 contains procedures for computations with vector space basis of a factor algebra including finiteness check and dimension computation.

The Section 7.10.3 [fpaprops_lib], page 659 contains procedures for determining important ring-theoretic properties including Gelfand-Kirillov dimension.

The Section 7.10.2 [fpalgebras_lib], page 645 contains procedures for the generation of various algebras, including group algebras of finitely presented groups in the Letterplace ring.

The Section 7.5.12 [ncfactor_lib], page 486 contains the procedure `ncfactor` for factorizing polynomials in the Letterplace ring.

See Section 7.3.2 [bracket], page 335; Section 5.1.88 [maxideal], page 219; Section 7.8.9 [reduce (letterplace)], page 631; Section 7.8.10 [rightstd (letterplace)], page 632; Section 7.8.11 [std (letterplace)], page 632; Section 7.8.14 [twostd (letterplace)], page 634.

7.7.4 References and history of Letterplace

Letterplace has undergone several stages of development.

The first one, the pure Letterplace implementation for homogeneous ideals, was created by V. Levandovskyy and H. Schoenemann in 2007-2009.

Later in 2010-2014, experiments with advanced (among other, with shift-invariant) data structures were performed by V. Levandovskyy, B. Schnitzler and G. Studzinski, and new libraries for K -dimension, K -bases, and Ufnarovskij graph were written.

The next stage started in 2017, when K. Abou Zeid joined the team of H. Schoenemann and V. Levandovskyy. Those recent activities led to the change of interface to the one, usual in the free algebra. The Letterplace data structure is still at heart of the implementation, though not explicitly visible by default. It has been generalized to support Z as coefficient ring (together with T. Metzlaff (RWTH Aachen and INRIA Sophia Antipolis)); to support bimodules and compute syzygies and lifts, to name a few. We are grateful to L. Schmitz (RWTH Aachen) for his contributions to the development.

References:

[LL09]: Roberto La Scala and Viktor Levandovskyy, "Letterplace ideals and non-commutative Groebner bases", Journal of Symbolic Computation, Volume 44, Issue 10, October 2009, Pages 1374-1393, see `http://dx.doi.org/10.1016/j.jsc.2009.03.002`.

[LL13]: Roberto La Scala and Viktor Levandovskyy, "Skew polynomial rings, Groebner bases and the letterplace embedding of the free associative algebra", Journal of Symbolic Computation, Volume 48, Issue 1, January 2013, Pages 1374-1393, see `http://dx.doi.org/10.1016/j.jsc.2012.05.003` and also `http://arxiv.org/abs/1009.4152`.

[LSS13]: Viktor Levandovskyy, Grischa Studzinski and Benjamin Schnitzler , "Enhanced Computations of Groebner Bases in Free Algebras as a New Application of the Letterplace Paradigm", Proc. ISSAC 2013, ACM Press, 259-266, see `https://doi.org/10.1145/2465506.2465948`.

[L14]: Roberto La Scala, "Extended letterplace correspondence for nongraded noncommutative ideals and related algorithms", International Journal of Algebra and Computation, Volume 24, Number 08, Pages 1157-1182, 2014, see also `https://doi.org/10.1142/S0218196714500519`.

[Mora16]: Teo Mora, "Solving Polynomial Equation Systems IV: Volume 4, Buchberger Theory and Beyond.", Cambridge University Press, 2016.

[LMZ20]: Viktor Levandovskyy, Tobias Metzlaff and Karim Abou Zeid, "Computation of free non-commutative Groebner Bases over Z with SINGULAR:LETTERPLACE", Proc. ISSAC 2020, Pages 312-319, ACM Press (2020), `https://dl.acm.org/doi/10.1145/3373207.3404052`. Video of the talk is at `https://av.tib.eu/media/50124`.

[LSZ20]: Viktor Levandovskyy, Hans Schoenemann and Karim Abou Zeid, "LETTERPLACE - a Subsystem of SINGULAR for computations with free algebras via Letterplace Embedding", Proc. ISSAC 2020, 305-311, ACM Press, `https://dl.acm.org/doi/10.1145/3373207.3404056`. Video of the talk is at `https://av.tib.eu/media/50123`.

[SL20]: Leonard Schmitz and Viktor Levandovskyy : Formally Verifying Proofs for Algebraic Identities of Matrices . In: Intelligent Computer Mathematics (Proceedings of the CICM 2020), Pages 222-236, Springer LNAI, LNCS (2020).

7.8 Functions (letterplace)

This chapter gives a complete reference of all functions and commands of the LETTERPLACE kernel, i.e. all built-in commands (for the numerous LETTERPLACE libraries see Section 7.10 [LETTERPLACE libraries], page 639).

The general syntax of a function is

[target =] function_name (<arguments>);

Note, that both **Control structures** and **System variables** of LETTERPLACE are the same as of SINGULAR (see Section 5.2 [Control structures], page 290, Section 5.3 [System variables], page 302).

7.8.1 dim (letterplace)

Syntax: `dim(` ideal_expression`)`

Type: int

Purpose: Compute the Gelfand-Kirillov dimension of the algebra basering/(input ideal). Uses Ufnarovskij graph for computations.

Note: the input ideal must be given as a two-sided Groebner basis.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r, 5);
ideal I = z;
dim(twostd(I)); // GK dimension is infinite
↦ -1
I = x,y,z;
dim(twostd(I));
↦ 0
```

```
I = x*y, x*z, z*y, z*z;
dim(twostd(I));
↦ 2
I = y*x - x*y, z*x - x*z, z*y - y*z;
I = twostd(I); I;
↦ I[1]=z*y-y*z
↦ I[2]=z*x-x*z
↦ I[3]=y*x-x*y
dim(I); // 3, as expected for R/I = K[x,y,z]
↦ 3
```

See Section 7.10.1 [fpadim_lib], page 639.

7.8.2 fetch (letterplace)

Syntax: `fetch` (ring_name, name)
`fetch` (ring_name, name, intvec_expression)

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: maps objects between rings. `fetch` is the identity map between rings and qrings, in the first case the i-th variable of the source ring is mapped to the i-th variable of the basering. If the basering has less variables than the source ring these variables are mapped to zero. The intvec in the 3rd argument describes the permutation of the variables: an i at position j maps the variable `var(j)` of the source to the variable `var(i)` of the destination.
A zero means that that variable/parameter is mapped to 0.
The coefficient fields must be compatible. (See Section 4.12 [map], page 106 for a description of possible mappings between different ground fields).
`fetch` offers a convenient way to change variable names or orderings, or to map objects from a ring to a factor ring of that ring or vice versa.
`option(Imap);` reports the mapping.

Note: Compared with `imap`, `fetch` uses the position of the ring variables, not their names.

Example:

```
  LIB "freegb.lib";
  ring r = (0,a),(x,y,z),dp;
  ring R = freeAlgebra(r,4,2); // free bimodule of rank 2
  poly p = z^2/a - a*y;
  ideal I = x,y,z,a*z*y*x - x*y + 7;
  module M = (x*y*a +3)*ncgen(1)*gen(1), ncgen(2)*gen(2)*z, ncgen(2)*gen(2
  M; // note that a stands on the left
↦ M[1]=(a)*x*y*ncgen(1)*gen(1)+3*ncgen(1)*gen(1)
↦ M[2]=ncgen(2)*z*gen(2)
↦ M[3]=(a)*ncgen(2)*x*y*gen(2)-7*ncgen(2)*gen(2)
  ring r2 = 0,(a,z,y,x),dp; // note: a is a variable in r2
  ring R2 = freeAlgebra(r2,6,2);
  fetch(R,p); // correctly processes incorrect input
↦ // ** Not defined: Cannot map a rational fraction and make a polynomial
  ut of it! Ignoring the denominator.
↦ y*y
  fetch(R,I);
```

```
↦ _[1]=a
↦ _[2]=z
↦ _[3]=y
↦ _[4]=-a*z+7
  fetch(R,M);
↦ _[1]=3*ncgen(1)*gen(1)
↦ _[2]=ncgen(2)*y*gen(2)
↦ _[3]=-7*ncgen(2)*gen(2)
  setring R; // now we show the factor ring behavior
  ideal J = y*x-x*y,z; J = twostd(J); J;
↦ J[1]=z
↦ J[2]=y*x-x*y
  qring Q = J;
  fetch(R,p);
↦ 1/(a)*z*z+(-a)*y
  NF(_, twostd(0)); // the canonical representative in Q
↦ (-a)*y
  fetch(R,I);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦ _[4]=(a)*z*y*x-x*y+7
  NF(_, twostd(0)); // the canonical representative in Q
↦ _[1]=x
↦ _[2]=y
↦ _[3]=0
↦ _[4]=-x*y+7
  fetch(R,M);
↦ _[1]=(a)*x*y*ncgen(1)*gen(1)+3*ncgen(1)*gen(1)
↦ _[2]=ncgen(2)*z*gen(2)
↦ _[3]=(a)*ncgen(2)*x*y*gen(2)-7*ncgen(2)*gen(2)
  NF(_, twostd(0)); // the canonical representative in Q
↦ _[1]=(a)*x*y*ncgen(1)*gen(1)+3*ncgen(1)*gen(1)
↦ _[2]=0
↦ _[3]=(a)*ncgen(2)*x*y*gen(2)-7*ncgen(2)*gen(2)
```

See Section 7.8.4 [imap (letterplace)], page 627; Section 4.12 [map], page 106; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127.

7.8.3 freeAlgebra (letterplace)

Syntax:

`freeAlgebra(` ring_expression r, int_expression d)

Type: ring

Purpose: Creates a free (letterplace) ring with the variables of the ring `r` up to the degree (length) bound `d`, with the monomial ordering, determined by those on the ring `r`.

Note: A letterplace ring has an attribute called `isLetterplaceRing`, which is zero for non-letterplace rings and contains the number of variables of the free algebra it encodes, otherwise.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r, 7); // this ordering is degree right lex
R;
↦ // coefficients: QQ
↦ // number of vars : 21
↦ //        block   1 : ordering dp
↦ //                  : names    x y z x y z x y z x y z x y z x y z x y z
↦ //        block   2 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
attrib(R,"isLetterplaceRing");
↦ 3
ring r2 = 0,(x,y,z),lp;
ring R2 = freeAlgebra(r2, 5); // note, that this ordering is NOT left or ri
R2;
↦ // coefficients: QQ
↦ // number of vars : 15
↦ //        block   1 : ordering a
↦ //                  : names    x y z x y z x y z x y z x y z
↦ //                  : weights  1 0 0 1 0 0 1 0 0 1 0 0 1 0 0
↦ //        block   2 : ordering a
↦ //                  : names    x y z x y z x y z x y z x y z
↦ //                  : weights  0 1 0 0 1 0 0 1 0 0 1 0 0 1 0
↦ //        block   3 : ordering a
↦ //                  : names    x y z x y z x y z x y z x y z
↦ //                  : weights  0 0 1 0 0 1 0 0 1 0 0 1 0 0 1
↦ //        block   4 : ordering lp
↦ //                  : names    x y z x y z x y z x y z x y z
↦ //        block   5 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
attrib(R2,"isLetterplaceRing");
↦ 3
```

See Section 7.9.2 [Monomial orderings on free algebras], page 636.

7.8.4 imap (letterplace)

Syntax: `imap` (ring_name, name)

Type: number, poly, vector, ideal, module, matrix or list (the same type as the second argument)

Purpose: identity map on common subrings. `imap` is the map between rings and qrings with compatible ground fields which is the identity on variables and parameters of the same name and 0 otherwise. (See Section 4.12 [map], page 106 for a description of possible mappings between different ground fields). Useful for embeddings as well as for mappings from/to rings with/without parameters. Compared with `fetch`, `imap` uses the names of variables and parameters. Unlike `map` and `fetch`, `imap` can map parameters to variables, by forgetting the commutativity of parameters with each other and with variables.
Mapping rational functions which are not polynomials to polynomials is undefined.

Example:

```
ring r = (0,a),(x,y,z),dp;
```

```
  LIB "freegb.lib";
  ring R = freeAlgebra(r,4,2); // free bimodule of rank 2
  poly p = z^2/a - a*y;
  ideal I = x,y,z,a*z*y*x - x*y + 7;
  module M = (x*y*a +3)*ncgen(1)*gen(1), ncgen(2)*gen(2)*z, ncgen(2)*gen(2
  M; // note that a stands on the left
↦ M[1]=(a)*x*y*ncgen(1)*gen(1)+3*ncgen(1)*gen(1)
↦ M[2]=ncgen(2)*z*gen(2)
↦ M[3]=(a)*ncgen(2)*x*y*gen(2)-7*ncgen(2)*gen(2)
  ring r2 = 0,(a,z,y,x),dp; // note: a is a variable in r2
  ring R2 = freeAlgebra(r2,6,2);
  imap(R,p); // correctly processes incorrect input
↦ // ** Not defined: Cannot map a rational fraction and make a polynomial
   ut of it! Ignoring the denominator.
↦ z*z-a*y
  imap(R,I);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦ _[4]=a*z*y*x-x*y+7
  imap(R,M);
↦ _[1]=a*x*y*ncgen(1)*gen(1)+3*ncgen(1)*gen(1)
↦ _[2]=ncgen(2)*z*gen(2)
↦ _[3]=a*ncgen(2)*x*y*gen(2)-7*ncgen(2)*gen(2)
```

See Section 7.8.2 [fetch (letterplace)], page 625; Section 4.12 [map], page 106; Section 4.20.1 [qring], page 127; Section 4.20 [ring], page 127.

7.8.5 lift (letterplace)

Syntax: `lift (` ideal_expression`,` subideal_expression `)`
`lift (` module_expression`,` submodule_expression `)`

Type: matrix

Purpose: computes the transformation matrix which expresses the generators of a subbimodule in terms of the generators of a bimodule.
More precisely, if `m` is the module (or ideal), `sm` the submodule (or ideal), and `T` the transformation matrix returned by lift, then the substitution of each `ncgen(i)` in `T` by the `m[i]` delivers a matrix, say `N`. The `i`-th generator of `sm` is equal to the sum of elements in the `i`-th column of `N`.

Note: Gives a warning if `sm` is not a submodule.

Note: The procedure Section 7.10.4.12 [testLift], page 671 can be used for testing the result.

Example:

```
  LIB "freegb.lib";
  ring r = 0,(x,y),(c,Dp);
  ring R = freeAlgebra(r, 7, 2);
  ideal I = std(x*y*x + 1);
  print(matrix(I));
↦ x*y-y*x,y*x*x+1
  ideal SI = x*I[1]*y + y*x*I[2], I[1]*y*x + I[2]*y;
  matrix T = lift(I, SI);
```

```
print(T);
↦ y*ncgen(1)*x*x+x*ncgen(1)*y,y*x*ncgen(1)+y*ncgen(1)*x+ncgen(1)*y*x,
↦ y*ncgen(2)*x,                y*ncgen(2)
print(matrix(SI)); // the original generators
↦ y*x*y*x*x+x*x*y*y-x*y*x*y+y*x,x*y*y*x+y*x*x*y-y*x*y*x+y
print(matrix(testLift(I,T))); // test for the result of lift
↦ y*x*y*x*x+x*x*y*y-x*y*x*y+y*x,x*y*y*x+y*x*x*y-y*x*y*x+y
```

See Section 4.6 [ideal], page 80; Section 7.8.6 [liftstd (letterplace)], page 629; Section 7.8.13 [syz (letterplace)], page 633; Section 7.8.14 [twostd (letterplace)], page 634.

7.8.6 liftstd (letterplace)

Syntax: `liftstd` (ideal_expression`,` matrix_name)
`liftstd` (module_expression`,` matrix_name)
`liftstd` (ideal_expression`,` matrix_name`,` module_name)
`liftstd` (module_expression`,` matrix_name`,` module_name)

Type: ideal or module

Purpose: returns a Groebner basis of a two-sided ideal or a bimodule and the transformation matrix from the given ideal, resp. module, to the Groebner basis from the output.
That is, if `m` is the module (or ideal), `sm` the submodule (or ideal), and `T` the transformation matrix returned by lift, then the substitution of each `ncgen(i)` in `T` by the `m[i]` delivers a matrix, say `N`. The `i`-th generator of `sm` is equal to the sum of elements in the `i`-th column of `N`.
In an optional third argument the syzygy bimodule will be returned.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),(c,Dp);
ring R = freeAlgebra(r, 8, 2);
ideal I = x*y*x + 1;
matrix T; module S;
ideal SI = liftstd(I,T,S);
print(matrix(SI));
↦ x*y-y*x,y*x*x+1
print(matrix(testLift(I,T))); // test for the result of lift
↦ x*y-y*x,y*x*x+1
S; // the bisyzygy module of I
↦ S[1]=[x*y*ncgen(1)*x*y+y*x*x*y*ncgen(1)-y*x*ncgen(1)*y*x-ncgen(1)*y*x*x*
   +y*ncgen(1)-ncgen(1)*y]
↦ S[2]=[x*x*y*ncgen(1)*x-x*ncgen(1)*y*x*x-x*ncgen(1)+ncgen(1)*x]
↦ S[3]=[x*y*ncgen(1)*x*x*y+y*x*x*x*y*ncgen(1)-y*x*x*ncgen(1)*y*x-ncgen(1)*
   *x*x*x*y+x*y*ncgen(1)+y*x*ncgen(1)-ncgen(1)*x*y-ncgen(1)*y*x]
↦ S[4]=[x*y*x*y*ncgen(1)*x-ncgen(1)*y*x*y*x*x+y*ncgen(1)*x-ncgen(1)*y*x]
testSyz(I,S);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
```

See Section 4.6 [ideal], page 80; Section 7.8.5 [lift (letterplace)], page 628; Section 7.8.13 [syz (letterplace)], page 633; Section 7.8.14 [twostd (letterplace)], page 634.

7.8.7 modulo (letterplace)

Syntax:

Syntax: `modulo` (ideal_expression, ideal_expression)
`modulo` (module_expression, module_expression)

Type: module

Purpose: computes the kernel of the bimodule homomorphism from the free bimodule (determined in basering) to its factor-bimodule modulo the second argument. The first argument determines the homomorphism via images of the canonical free bimodule generators.
If `option(returnSB)` is set, a Groebner basis is returned, otherwise a generating set.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r,7,2); // free bimodule of rank 2
ideal I = x*y*z - z*y*x;
I = twostd(I); I;
↦ I[1]=z*y*x-x*y*z
modulo(y,twostd(0)); // shows the canonical generator of the kernel
↦ _[1]=ncgen(1)*y*gen(1)-y*ncgen(1)*gen(1)
// which can be interpreted as (1 otimes y - y otimes 1)
module M = modulo(y, I);
print(M); // as we see (z E y - y E z) generates the kernel
↦ ncgen(1)*y-y*ncgen(1),z*ncgen(1)*x-x*ncgen(1)*z
// of bimodule homomorphism sending E to y
```

See Section 4.6 [ideal], page 80; Section 7.8.5 [lift (letterplace)], page 628; Section 7.8.6 [liftstd (letterplace)], page 629; Section 4.14 [module], page 112; Section 7.8.8 [ncgen], page 630; Section 5.1.111 [option], page 234; Section 7.8.13 [syz (letterplace)], page 633.

7.8.8 ncgen

Syntax: `ncgen` (int_expression)

Type: poly

Purpose: returns the i-th free non-commutative generator of a free bimodule.

Note: `ncgen` in bimodules is used together with the commutative `gen`, which encodes the component or the position in a vector.

Example:

```
  LIB"freegb.lib";
  ring r= 0,(x,y),dp;
  ring R = freeAlgebra(r,4,3); // R supports free bimodule up to rank 3
  typeof(ncgen(2));
↦ poly
  vector v = [ncgen(1)*x, x*ncgen(2), y*ncgen(3)*x];
  v;
↦ y*ncgen(3)*x*gen(3)+ncgen(1)*x*gen(1)+x*ncgen(2)*gen(2)
  print(v*x);
↦ [ncgen(1)*x*x,x*ncgen(2)*x,y*ncgen(3)*x*x]
```

```
  print(x*v);
↦ [x*ncgen(1)*x,x*x*ncgen(2),x*y*ncgen(3)*x]
```

See Section 7.8.3 [freeAlgebra (letterplace)], page 626; Section 4.14 [module], page 112; Section 4.23 [vector], page 134.

7.8.9 reduce (letterplace)

Syntax:
`reduce` (poly_expression, ideal_expression)
`reduce` (poly_expression, ideal_expression, int_expression)
`reduce` (vector_expression, ideal_expression)
`reduce` (vector_expression, ideal_expression, int_expression)
`reduce` (vector_expression, module_expression, int_expression)
`reduce` (ideal_expression, ideal_expression)
`reduce` (ideal_expression, ideal_expression, int_expression)

Type: the type of the first argument

Purpose: reduces a polynomial, vector, or ideal (the first argument) to its **two-sided** normal form with respect to the second argument, meant to be an ideal, represented by its two-sided Groebner basis (otherwise, the result may have no meaning).
returns 0 if and only if the polynomial (resp. vector, ideal) is an element (resp. subideal) of the ideal.
The third (optional) argument of type int modifies the behavior:

0 default

1 consider only the leading term and do no tail reduction.

2 tail reduction: in the local/mixed ordering case: reduce also with bad ecart

4 reduce without division, return possibly a non-zero constant multiple of the remainder

Note: The commands `reduce` and `NF` are synonymous.

Note: A two-sided Groebner presentation of a polynomial with respect to a two-sided ideal can be computed by the procedure Section 7.10.4.5 [lpDivision], page 666 from Section 7.10.4 [freegb_lib], page 665.

Example:

```
  LIB "freegb.lib";
  ring r = 0,(x,y),dp;
  ring R = freeAlgebra(r,5);
  ideal I = x*x + y*y - 1; // 2D sphere
  ideal J = twostd(I); // computes a two-sided Groebner basis
  J; // it is finite and nice
↦ J[1]=x*x+y*y-1
↦ J[2]=y*y*x-x*y*y
  poly g = x*y*y - y*y*x;
  reduce(g,J); // 0, hence g belongs to J
↦ 0
  poly h = x*y*y*x - y*x*x;
  reduce(h,J); // the rest of two-sided division of h by J
↦ -y*y*y*y+y*y*y+y*y-y
  qring Q = J; // switch to K<x,y>/J
  reduce(x*y*y - y*y*x,twostd(0)); //image of g above
↦ 0
```

```
reduce(x*y*y*x - y*x*x,std(0)); //image of h above
↦ -y*y*y*y+y*y*y+y*y-y
```

See also Section 7.8.5 [lift (letterplace)], page 628.

7.8.10 rightstd (letterplace)

Syntax: `rightstd(` ideal_expression`);` `rightstd(` module_expression`);`

Type: ideal or module

Purpose: Compute a right Groebner basis of the set of generators of the input ideal/module.

Note: It is also effective in factor rings.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,z),dp;
ring R = freeAlgebra(r,7);
ideal I = z, x*z, x*x*z;
rightstd(I); // a right GB of I in K<x,z>
↦ _[1]=z
↦ _[2]=x*z
↦ _[3]=x*x*z
qring Q = twostd(x*z); // now we change to the factor algebra modulo x*z
ideal I = imap(R,I);
rightstd(I); // a right GB in a factor algebra
↦ _[1]=z
reduce(I,twostd(0)); // an explanation for the latter
↦ _[1]=z
↦ _[2]=0
↦ _[3]=0
```

7.8.11 std (letterplace)

Syntax: `std(` ideal_expression`);` `std(` module_expression`);`

Type: ideal or module

Purpose: Alias to Section 7.8.14 [twostd (letterplace)], page 634.

7.8.12 subst (letterplace)

Syntax:
`subst (` poly_expression`,` variable`,` poly_expression `)`
`subst (` poly_expression`,` variable`,` poly_expression `,...` variable`,` poly_expression `)`
`subst (` vector_expression`,` variable`,` poly_expression `)`
`subst (` ideal_expression`,` variable`,` poly_expression `)`
`subst (` module_expression`,` variable`,` poly_expression `)`

Type: poly, vector, ideal or module (corresponding to the first argument)

Purpose: substitutes one or more ring variable(s)/parameter variable(s) by (a) polynomial(s). Note that in the case of more than one substitution pair, the substitutions will be performed sequentially and not simultaneously. The below examples illustrate this behaviour.
Note, that the coefficients must be polynomial when substituting a parameter.

Note: When dealing with free non-commutative bimodules, their generators `ncgen(i)` can be used as variables in `subst` and therefore substituted in the corresponding vector component `gen(i)`.

Example:

```
  LIB "freegb.lib";
  ring r = 0,(x,y,z),dp;
  ring R = freeAlgebra(r,5,2);
  poly p = z^2 - y*x;
  subst(p, x, -y);
↦ y*y+z*z
  ideal I = z*y*x - x*y*z;
  subst(I, y, p);
↦ _[1]=-z*y*x*x+z*z*z*x+x*y*x*z-x*z*z*z
  subst(I, x, z); // produces zero
↦ _[1]=0
  module M = I*ncgen(1)*gen(1), ncgen(1)*gen(1)*I, I*ncgen(2)*gen(2);
  M;
↦ M[1]=z*y*x*ncgen(1)*gen(1)-x*y*z*ncgen(1)*gen(1)
↦ M[2]=ncgen(1)*z*y*x*gen(1)-ncgen(1)*x*y*z*gen(1)
↦ M[3]=z*y*x*ncgen(2)*gen(2)-x*y*z*ncgen(2)*gen(2)
  subst(M, x, z); // produces zero
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
  subst(M, ncgen(2), z); // evaluates ncgen(2) at z, see the 2nd component
↦ _[1]=z*y*x*ncgen(1)*gen(1)-x*y*z*ncgen(1)*gen(1)
↦ _[2]=ncgen(1)*z*y*x*gen(1)-ncgen(1)*x*y*z*gen(1)
↦ _[3]=z*y*x*z*gen(2)-x*y*z*z*gen(2)
```

See Section 4.6 [ideal], page 80; Section 4.12 [map], page 106; Section 4.14 [module], page 112.

7.8.13 syz (letterplace)

Syntax: syz (ideal_expression)
syz (module_expression)

Type: module

Purpose: computes the first syzygy (i.e., the module of relations of the given generators) bimodule of the ideal, resp. module.
If `option(returnSB)` is set, a Groebner basis is returned, otherwise a generating set.

Example:

```
  LIB "freegb.lib";
  ring r = 0,(x,y),(c,Dp);
  ring R = freeAlgebra(r, 7, 2);
  ideal I = twostd(x*y*x + 1);
  I;
↦ I[1]=x*y-y*x
↦ I[2]=y*x*x+1
  module S = syz(I);
  print(S);
↦ ncgen(1)*x*x,S[1,2],S[1,3],S[1,4],S[1,5],
↦ S[2,1],      S[2,2],S[2,3],S[2,4],S[2,5]
```

```
  testSyz(I,S);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
```

See Section 4.6 [ideal], page 80; Section 7.8.5 [lift (letterplace)], page 628; Section 7.8.6 [liftstd (letterplace)], page 629; Section 4.14 [module], page 112; Section 7.8.8 [ncgen], page 630; Section 5.1.111 [option], page 234.

7.8.14 twostd (letterplace)

Syntax: `twostd(` ideal_expression`);` `twostd(` module_expression`);`

Type: ideal

Purpose: returns a two-sided Groebner basis of the two-sided ideal, generated by the input, which is treated as a set of two-sided generators.

Example:

```
LIB "freegb.lib";
ring r = 3,(x,d),dp; // notice: we work over Z/3Z
ring R = freeAlgebra(r,5);
ideal I = x^4, d^3, d*x - x*d - 1;
twostd(I); // a proper ideal, note x^3 as a generator
↦ _[1]=d*x-x*d-1
↦ _[2]=d*d*d
↦ _[3]=x*x*x
ideal J = x^2, d^3, d*x - x*d - 1;
twostd(J); // the whole ring
↦ _[1]=1
ideal T = twostd(ideal(d*x - x*d - 1));
T;
↦ T[1]=d*x-x*d-1
qring Q = T; // thus Q is the Weyl algebra over Z/3z
ideal I = x^4, d^3;
twostd(I);
↦ _[1]=d*d*d
↦ _[2]=x*x*x
ideal J = x^2, d^3;
twostd(J);
↦ _[1]=1
```

See Section 4.6 [ideal], page 80; Section 7.8.5 [lift (letterplace)], page 628; Section 7.8.6 [liftstd (letterplace)], page 629; Section 4.14 [module], page 112; Section 7.8.8 [ncgen], page 630; Section 5.1.111 [option], page 234; Section 7.8.10 [rightstd (letterplace)], page 632; Section 7.8.13 [syz (letterplace)], page 633.

7.8.15 vdim (letterplace)

Syntax: `vdim (` ideal_expression `)`

Type: int

Purpose: computes the vector space dimension respective to the ground field of the ring modulo the ideal, generated by the leading terms of the given generators. If the generators form a standard basis, this is the same as the vector space dimension of the ring, resp. free module, modulo the ideal, resp. module.
If the ideal is not finite dimensional over the ground field, -1 is returned.
The non-commutative analog of the Section 5.1.69 [kbase], page 205 command is Section 7.10.1.4 [lpMonomialBasis], page 642 from Section 7.10.1 [fpadim_lib], page 639.

Example:

```
  LIB "fpadim.lib";
  ring r = 0,(x,y),dp;
  ring R = freeAlgebra(r,5);
  ideal I = x*x + x, y*y+y, x*y*x + x;
  ideal G = twostd(I); G;
↦ G[1]=y*y+y
↦ G[2]=x*x+x
↦ G[3]=x*y*x+x
  vdim(G); // 6
↦ 6
  lpMonomialBasis(5,0,G); // lists the K-basis explicitly
↦ _[1]=1
↦ _[2]=x
↦ _[3]=y
↦ _[4]=y*x
↦ _[5]=x*y
↦ _[6]=y*x*y
```

See Section 7.8.1 [dim (letterplace)], page 624; Section 7.10.1 [fpadim_lib], page 639; Section 7.8.14 [twostd (letterplace)], page 634.

7.9 Mathematical background (letterplace)

7.9.1 Free associative algebras

Let V be a K -vector space, spanned by the symbols x_1 ,. . . , x_n . A free associative algebra in x_1 ,. . . , x_n over K , denoted by K

$< x_1$,. . . , $x_n >$

is also known as the tensor algebra $T(V)$ of V ; it is also the monoid K -algebra of the free monoid $< x_1$,. . . , $x_n >$. The elements of this free monoid constitute an infinite K -basis of K

$< x_1$,. . . , $x_n >$, where the identity element (the empty word) of the free monoid is identified with the 1 in K . Yet in other words, the monomials of K

$< x_1$,. . . , $x_n >$ are the words of finite length in the finite alphabet { x_1 ,. . . , x_n }.

The algebra K

$< x_1$,. . . , $x_n >$ is an integral domain, which is not (left, right, weak or two-sided) Noetherian for $n > 1$; hence, a Groebner basis of a finitely generated ideal might be infinite. Therefore, a general computation takes place **up to an explicit degree (length) bound**, provided by the user. The free associative algebra can be regarded as a graded algebra in a natural way.

Definition. An associative algebra A is called **finitely presented (f.p.)**, if it is isomorphic to

K

$< x_1$,. . . , $x_n > /I$, where I is a two-sided ideal.

A is called **standard finitely presented (s.f.p.)**, if there exists a monomial ordering, such that I is given via its **finite** Groebner basis G .

7.9.2 Monomial orderings on free algebras

We provide many types of orderings for non-commutative Groebner bases up to a degree (length) bound. In general it is not clear, whether a given generating set has a finite Groebner bases with respect to some ordering.

Let $X =$ { x_1 ,. . . , x_n } be a set of symbols. A total ordering `<` on the free monoid $< X >$ with 1 as the neutral element is called a **monomial ordering** if

it is a well-ordering, i.e., every non empty subset has a least element with respect to `<`, and

it is compatible with multiplication, that is $u < v$ implies $aub < avb$ for all u , v , a and b in $< X >$.

Note that the latter implies $1 <= m$ for all m in $< X >$.

The **left lexicographical ordering** on $< X >$ with $x_1 > x_2 > \ldots > x_n$ is defined as follows: For arbitrary a , b in $< X >$ we say that $a < b$, if

$\exists u \in \langle X\rangle \setminus \{1\} :\ au = b$ or

$\exists u,\ v,\ w \in \langle X\rangle\ \exists x_i,\ x_j \in X : a = ux_iv,\ b = ux_jw$ and $i < j$ holds.

Note: left lex is **not** a monomial ordering, though it is a natural choice to break ties after, say, comparing elements by the total degree.

In a similar manner one can define the **right lexicographical ordering**.

On the monoid $(N_0, +)$ define the **weight** homomorphism $w : \langle X\rangle \to N_0$, uniquely determined by $w(x_i) = w_i$ in N_0 for $1 <= i <= n$.

As a special case, define the **length** len:$\langle X\rangle \to N_0$ by $len(x_i) = 1$ for $1 <= i <= n$.

For any ordering `<<` on $< X >$ and any weight $w : \langle X\rangle \to N_0$ define an ordering $<$, called the w **-weight extension of** $<<$ as follows: For arbitrary a , b in $< X >$ we say that $a < b$ if

$w(a) < w(b)$ or

$w(a) = w(b)$ and $a << b$ holds.

An ordering `<` on $< X >$ **eliminates** a certain subset $\emptyset \neq Y \subset X$ if for all $f \in K\langle X\rangle \setminus \{0\}$ one has $lm(f) \in K\langle X \setminus Y\rangle \Rightarrow f \in K\langle X \setminus Y\rangle \subseteq K\langle X\rangle$.

In a ring declaration, Letterplace supports the following monomial orderings.

We illustrate each of the available choices by an example on the free monoid $< x_1 , x_2 , x_3 >$, where we order the monomials

$x_1x_1x_1$, $x_3x_2x_1$, $x_1x_2x_3$, $x_3x_3x_3$, x_3x_1 , x_2x_2 , x_1x_3 , x_2x_3 , x_1 , x_2 and x_3 correspondingly.

‘`dp`’ The **degree right lexicographical ordering** is the length-weight extension of the right lexicographical ordering.

With respect to the ordering ‘`dp`’, the test monomials are ordered as follows:

$x_1x_1x_1 > x_3x_2x_1 > x_1x_2x_3 > x_3x_3x_3 > x_3x_1 > x_2x_2 > x_1x_3 > x_2x_3 > x_1 > x_2 > x_3$

‘`Dp`’ The **degree left lexicographical ordering** is the length-weight extension of the left lexicographical ordering.

With respect to the ordering ‘`Dp`’, the test monomials are ordered as follows:

$x_1x_1x_1 > x_1x_2x_3 > x_3x_2x_1 > x_3x_3x_3 > x_1x_3 > x_2x_2 > x_2x_3 > x_3x_1 > x_1 > x_2 > x_3$

‘Wp(w) for intvec w’

The **weighted degree left lexicographical ordering** is the w -weight extension of the left lexicographical ordering with weight $w : \langle X \rangle \to N_0$ uniquely determined by strict positive $w(x_i) = w_i > 0$.

With respect to the ordering ‘Wp(1, 2, 1)’, the test monomials are ordered as follows:

$x_1x_2x_3 > x_2x_2 > x_3x_2x_1 > x_1x_1x_1 > x_2x_3 > x_3x_3x_3 > x_1x_3 > x_2 > x_3x_1 > x_1 > x_3$

‘lp’

Let $w^{(i)}$ be weights uniquely determined by $w^{(i)}(x_j) = \delta_{i,j}$ for $1 \leq i,j \leq n$ where δ denotes the Kronecker delta. Let $<_n$ be the $w^{(n)}$-weight extension of the left lexicographical ordering on $\langle X \rangle$ and inductively $<_i$ be the $w^{(i)}$-weight extension of $<_{i+1}$ for all $1 \leq i < n$. The monomial ordering lp corresponds to $<_1$ and eliminates $x_1, ..., x_j$ for all $1 \leq j < n$. We refer to it as to **left elimination ordering**.

The monomial ordering ‘lp’ corresponds to $<_1$ and eliminates { x_1 ,. . . , x_j } for all 1 <= j < n . We refer to it as to **left elimination ordering**.

With respect to the ordering ‘lp’, the test monomials are ordered as follows:

$x_1x_1x_1 > x_1x_2x_3 > x_3x_2x_1 > x_1x_3 > x_3x_1 > x_1 > x_2x_2 > x_2x_3 > x_2 > x_3x_3x_3 > x_3$

‘rp’

Let $w^{(i)}$ be weights uniquely determined by $w^{(i)}(x_j) = \delta_{i,j}$ for $1 \leq i,j \leq n$ where δ denotes the Kronecker delta. Let $<_1$ be the $w^{(1)}$-weight extension of the left lexicographical ordering on $\langle X \rangle$ and inductively $<_i$ be the $w^{(i)}$-weight extension of $<_{i-1}$ for all $1 < i \leq n$. The monomial ordering rp corresponds to $<_n$ and eliminates $\{x_j, \ldots, x_n\}$ for all $1 < j \leq n$. We refer to it as to **right elimination ordering**.

The monomial ordering ‘rp’ corresponds to $<_n$ and eliminates { x_j ,. . . , x_n } for all $1 < j <= n$. We refer to it as to **right elimination ordering**.

With respect to the ordering ‘rp’, the test monomials are ordered as follows:

$x_3x_3x_3 > x_1x_2x_3 > x_3x_2x_1 > x_2x_3 > x_1x_3 > x_3x_1 > x_3 > x_2x_2 > x_2 > x_1x_1x_1 > x_1$

‘(a(v), ordering) for intvec v’

For weight $v : \langle X \rangle \to N_0$ determined by $v(x_i) = v_i \in N_0$ with $1 \leq i \leq n$ and monomial ordering $\prec$ on $\langle X \rangle$, the *v-weight extension* of $\prec$ corresponds to (a(v), o). As a choice for $\prec$ there are currently two options implemented, which are dp and Dp. Notice that this ordering eliminates $\{x_i \in X \mid v(x_i) \neq 0\}$.

With respect to the ordering ‘(a(1, 0, 0), Dp)’, the test monomials are ordered as follows:

$x_1x_1x_1 > x_1x_2x_3 > x_3x_2x_1 > x_1x_3 > x_3x_1 > x_1 > x_3x_3x_3 > x_2x_2 > x_2x_3 > x_2 > x_3$

With ordering ‘(a(1, 1, 0), Dp)’ one obtains:

$x_1x_1x_1 > x_1x_2x_3 > x_3x_2x_1 > x_2x_2 > x_1x_3 > x_2x_3 > x_3x_1 > x_1 > x_2 > x_3x_3x_3 > x_3$

The examples are generated by the following code but with customized orderings denoted above.

```
LIB "freegb.lib";
ring r = 0, (x1,x2,x3),Dp; // variate ordering here
ring R = freeAlgebra(r, 4);
poly wr = x1*x1*x1+x3*x3*x3+x1*x2*x3+x3*x2*x1+x2*x2+x2*x3+x1*x3+x3*x1+x1+x2+x3;
wr; // polynomial will be automatically ordered according to the ordering on R
↦ x1*x1*x1+x1*x2*x3+x3*x2*x1+x3*x3*x3+x1*x3+x2*x2+x2*x3+x3*x1+x1+x2+x3
```

7.9.3 Groebner bases for two-sided ideals in free associative algebras

We say that a monomial v divides (two-sided or bilaterally) a monomial w , if there exist monomials $p, s \in X$, such that $w = p \cdot v \cdot s$, in other words v is a subword of w .

Let $T := K\langle x_1, \ldots, x_n\rangle$ be the free algebra and $<$ be a fixed monomial ordering on T.

For a subset $G \subset K\langle x_1, \ldots, x_n\rangle$, define the **leading ideal of G to be the two-sided ideal $LM(G) = {}_T\langle \{lm(g) \mid g \in G \setminus \{0\}\} \rangle_T \subseteq T$.**

A subset $G \subset I$ is a (two-sided) Groebner basis for the ideal I with respect to $<$, if $LM(G) = LM(I)$.

That is $\forall f \in I \setminus \{0\}$ there exists $g \in G$, such that $lm(g)$ divides $lm(f)$.

The notion of **Groebner-Shirshov** basis applies to more general algebraic structures, but means the same as Groebner basis for associative algebras.

Suppose, that the weights of the ring variables are strictly positive. We can interpret these weights as defining a non-standard grading on the ring. If the set of input polynomials is weighted homogeneous with respect to the given weights of the ring variables, then computing up to a weighted degree (and thus, also length) bound d

results in the **truncated Groebner basis** $G(d)$. In other words, by trimming elements of degree exceeding d from the complete Groebner basis G , one obtains precisely $G(d)$.

In general, given a set $G(d)$, which is the result of Groebner basis computation up to weighted degree bound d , then it is the complete finite Groebner basis, if and only if $G(2d-1) = G(d)$ holds.

Note: If the set of input polynomials is **not** weighted homogeneous with respect to the weights of the ring variables, and a Groebner is **not** finite,

then actually not much can be said precisely on the properties of the given ideal. By increasing the length bound bigger generating sets will be computed, but in contrast to the weighted homogeneous case some polynomials in of small length first enter the basis after computing up to a much higher length bound.

7.9.4 Bimodules and syzygies and lifts

Let $A = K$

$< x_1 , \ldots , x_n >$ be the free algebra. A free bimodule of rank r over A is $Ae_1A \oplus \ldots \oplus Ae_rA$, where e_i are the generators of the free bimodule.

NOTE: these e_i are freely non-commutative with respect to elements of A except constants from the ground field K .

The free bimodule of rank 1 AeA surjects onto the algebra A itself. A two-sided ideal of the algebra A can be converted to a subbimodule of AeA .

The **syzygy bimodule** or even **module of bisyzygies** of the given finitely generated subbimodule $N = \langle g_1, \ldots, g_m\rangle \subset \bigoplus_{i=1}^r Ae_iA$ is the kernel of the natural homomorphism of A -bimodules $\bigoplus_{j=1}^m A\epsilon_jA \longrightarrow \bigoplus_{i=1}^r Ae_iA$, $\epsilon_j \mapsto g_j$, that is $\sum_{j=1}^m \sum_k \ell_{jk}\epsilon_j r_{jk} \mapsto \sum_{j=1}^m \sum_k \ell_{jk} g_j r_{jk}$.

The syzygy bimodule is in general not finitely generated. Therefore as a bimodule, both the set of generators of the syzygy bimodule and its Groebner basis are computed up to a specified length bound.

Given a subbimodule N of a bimodule M , the **lift(ing)** process returns a matrix, which encodes the expression of generators $N_1, \ldots, N_s$

in terms of generators of $M_1, \ldots, M_m$ like this: $N_i = \sum_{j=1}^m \sum_k \ell_{jk} M_j r_{jk} = \sum_{j=1}^m T_{ij} M_j$,

where T_ij are elements from the enveloping algebra $R\langle X\rangle \otimes R\langle X\rangle$, encoded as elements of the free bimodule of rank m , namely by using the non-commutative generators of the free bimodule which we call `ncgen`.

7.9.5 Letterplace correspondence

The name letteplace has been inspired by the work of Rota and, independently, Feynman.

Already Feynman and Rota encoded the monomials (words) of the free algebra $x_{i_1}x_{i_2}\dots x_{i_m} \in K\langle x_1,\dots,x_n\rangle$ via the double-indexed letterplace (that is encoding the letter (= variable) and its place in the word) monomials $x(i_1|1)x(i_2|2)\dots x(i_m|m) \in K[X \times N]$, where $X = \{x_1,\dots,x_n\}$ and N is the semigroup of natural numbers, starting with 1 as the first possible place. Note, that the letterplace algebra $K[X \times N]$ is an infinitely generated commutative polynomial K-algebra. Since $K\langle x_1,\dots,x_n\rangle$ is not Noetherian, it is common to perform the computations with its ideals and modules up to a given degree bound.

Subject to the given degree (length) bound d, the truncated letterplace algebra $K[X \times (1, ..., d)]$ is finitely generated commutative polynomial K-algebra.

In [LL09] a natural shifting on letterplace polynomials was introduced and used. Indeed, there is 1-to-1 correspondence between two-sided ideals of a free algebra and so-called letterplace ideals in the letterplace algebra, see [LL09], [LL13], [LSS13] and [L14] for details. Note, that first this correspondence was established for graded ideals, but holds more generally for arbitrary ideals and subbimodules of a free bimodule of a finite rank. All the computations internally take place in the Letterplace algebra.

A letterplace monomial of length m is a monomial of a letterplace algebra, such that its m places are exactly 1,2,. . . , m . In particular, such monomials are multilinear with respect to places (i.e. no place, smaller than the length is omitted or filled more than with one letter). A letterplace polynomial is an element of the K -vector space, spanned by letterplace monomials. A letterplace ideal is generated by letterplace polynomials subject to two kind of operations:

the K-algebra operations of the letterplace algebra **and simultaneous shifting of places by any natural number n.**

Note: Letterplace correspondence naturally extends to the correspondence over

$R <$

x_1 ,. . . , x_n

$>$, where R is a commutative unital ring. The case $R = Z$ is implemented, in addition to

R being a field.

7.10 LETTERPLACE libraries

The content of libraries, created for Letterplace is described in the following subsections.

Use the LIB command for loading of single libraries.

See also Section 7.5.12 [ncfactor_lib], page 486 for the factorization of polynomials in noncommutative algebras.

7.10.1 fpadim_lib

Library: fpadim.lib

Purpose: Vector space dimension, basis and Hilbert series for finitely presented algebras (Letterplace)

Authors: Grischa Studzinski, grischa.studzinski at rwth-aachen.de
Viktor Levandovskyy, viktor.levandovskyy at math.rwth-aachen.de
Karim Abou Zeid, karim.abou.zeid at rwth-aachen.de

Support: Joint projects LE 2697/2-1 and KR 1907/3-1 of the Priority Programme SPP 1489: 'Algorithmische und Experimentelle Methoden in Algebra, Geometrie und Zahlentheorie' of the German DFG (2010-2013)
and Project II.6 of the transregional collaborative research centre SFB-TRR 195 'Symbolic Tools in Mathematics and their Application' of the German DFG (from 2017 on)

Note: - basering is a Letterplace ring
- all intvecs correspond to Letterplace monomials
- if a degree bound d is specified, d <= attrib(basering,uptodeg) holds

In the procedures below, 'iv' stands for intvec representation and 'lp' for the letterplace representation of monomials

Overview: Given the free associative algebra A = K<x_1,...,x_n> and a (finite or truncated) Groebner basis GB, one is interested in the following data:
- the K-dimension of A/<GB> (check for finiteness or explicit value) - the Hilbert series of A/<GB>
- the explicit monomial K-basis of A/<GB>
In order to determine these, we need
- the Ufnarovskij graph induced by GB
- the mistletoes of A/<GB> (which are special monomials in a basis)

The Ufnarovskij graph is used to determine whether A/<GB> has finite K-dimension. One has to check if the graph contains cycles. For the whole theory we refer to [Ufn]. Given a
reduced set of monomials GB one can define the basis tree, whose vertex set V consists of all normal monomials w.r.t. GB. For every two monomials m_1, m_2 in V there is a direct edge from m_1 to m_2, if and only if there exists x_k in {x_1,..,x_n}, such that m_1*x_k = m_2. The set M = {m in V | there is no edge from m to another monomial in V} is called the set of mistletoes. As one can easily see it consists of the endpoints of the graph. Since there is a unique path to every monomial in V, the whole graph can be described only from the knowledge of the mistletoes. Note that V corresponds to a basis of A/<GB>, so knowing the mistletoes we know a K-basis. The name mistletoes was given to those points because of these miraculous value and the algorithm is named sickle, because a sickle is the tool to harvest mistletoes. For more details see [Stu]. This package uses the Letterplace format introduced by [LL09]. The algebra can either be represented as a Letterplace ring or via integer vectors: Every variable will only be represented by its number, so variable one is represented as 1, variable two as 2 and so on. The monomial x_1*x_3*x_2 for example will be stored as (1,3,2). Multiplication is concatenation. Note that the approach in this library does not need an algorithm for computing the normal form. Note that fpa is an acronym for Finitely Presented Algebra.

References:

[Ufn] V. Ufnarovskij: Combinatorial and asymptotic methods in algebra, 1990. [LL09] R. La Scala, V. Levandovskyy: Letterplace ideals and non-commutative Groebner bases, Journal of Symbolic Computation, 2009. [Stu] G. Studzinski: Dimension computations in non-commutative, associative algebras, Diploma thesis, RWTH Aachen, 2010.

Procedures: See also: Section 7.10.3 [fpaprops_lib], page 659; Section 7.10.4 [freegb_lib], page 665; Section 7.10.5 [ncHilb_lib], page 672.

7.10.1.1 teach_lpKDimCheck

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: teach_lpKDimCheck(G);

Return: int, 1 if K-dimension of the factor algebra is infinite, 0 otherwise

Purpose: Checking a factor algebra for finiteness of the K-dimension

Assume: - basering is a Letterplace ring.

Example:

```
LIB "fpadim.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
setring R; // sets basering to Letterplace ring
ideal G = x*x, y*y,x*y*x;
// Groebner basis
ideal I = x*x, y*x*y, x*y*x;
// Groebner basis
teach_lpKDimCheck(G); // invokes procedure, factor algebra is of finite K-dimension
↦ 0
teach_lpKDimCheck(I); // invokes procedure, factor algebra is of infinite Kdimension
↦ 1
```

7.10.1.2 lpKDim

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: lpKDim(G); G an ideal in a letterplace ring

Return: int

Purpose: Computes the K-dimension of A/<G>
-1 means infinity

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Note: - Alias for vdim(G)

7.10.1.3 teach_lpKDim

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: teach_lpKDim(G[,degbound, n]); G an ideal, degbound, n optional integers

Return: int, the K-dimension of the factor algebra

Purpose: Compute the K-dimension of a factor algebra, given via an ideal

Assume: - basering is a Letterplace ring
- if you specify a different degree bound degbound,
degbound <= attrib(basering,uptodeg) holds.

Note: - If degbound is set, there will be a degree bound added. 0 means no
degree bound. Default: attrib(basering, uptodeg).
- n is the number of variables, which can be set to a different number.
Default: attrib(basering, lV).
- If the K-dimension is known to be infinite, a degree bound is needed

Example:

```
LIB "fpadim.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
setring R; // sets basering to Letterplace ring
ideal G = x*x, y*y,x*y*x;
// ideal G contains a Groebner basis
teach_lpKDim(G); //procedure invoked with ring parameters
↦ 6
// the factor algebra is finite, so the degree bound given by the Letterplace
// ring is not necessary
teach_lpKDim(G,0); // procedure without any degree bound
↦ 6
```

7.10.1.4 lpMonomialBasis

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: lpMonomialBasis(d, donly, J); d, donly integers, J an ideal

Return: ideal

Purpose: computes a list of free monomials in a Letterplace
basering R of degree at most d and not contained in <LM(J)>
if donly <> 0, only monomials of degree d are returned

Assume: - basering is a Letterplace ring.
- d <= attrib(basering,uptodeg) holds.
- J is a Groebner basis

Note: will be replaced with reduce(maxideal(d), J); soon

Example:

```
LIB "fpadim.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 7); setring R;
ideal J = x*y*x - y*x*y;
option(redSB); option(redTail);
J = letplaceGBasis(J);
J;
↦ J[1]=x*y*x-y*x*y
↦ J[2]=y*x*y*y*x-x*y*y*x*y
↦ J[3]=y*x*y*y*y*x-x*x*y*y*x*y
↦ J[4]=y*x*y*y*y*y*x-x*x*x*y*y*x*y
//monomials of degree 2 only in K<x,y>:
lpMonomialBasis(2,1,ideal(0));
↦ _[1]=x*x
↦ _[2]=y*x
↦ _[3]=x*y
↦ _[4]=y*y
//monomials of degree <=2 in K<x,y>
lpMonomialBasis(2,0,ideal(0));
↦ _[1]=1
↦ _[2]=x
↦ _[3]=y
↦ _[4]=x*x
↦ _[5]=y*x
↦ _[6]=x*y
↦ _[7]=y*y
//monomials of degree 3 only in K<x,y>/J
lpMonomialBasis(3,1,J);
↦ _[1]=x*x*x
↦ _[2]=y*x*x
↦ _[3]=y*y*x
↦ _[4]=x*x*y
↦ _[5]=y*x*y
↦ _[6]=x*y*y
↦ _[7]=y*y*y
//monomials of degree <=3 in K<x,y>/J
lpMonomialBasis(3,0,J);
↦ _[1]=1
↦ _[2]=x
↦ _[3]=y
↦ _[4]=x*x
↦ _[5]=y*x
↦ _[6]=x*y
↦ _[7]=y*y
↦ _[8]=x*x*x
↦ _[9]=y*x*x
↦ _[10]=y*y*x
↦ _[11]=x*x*y
↦ _[12]=y*x*y
↦ _[13]=x*y*y
↦ _[14]=y*y*y
```

7.10.1.5 lpHilbert

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: lpHilbert(G[,degbound,n]); G an ideal, degbound, n optional integers

Return: intvec, containing the coefficients of the Hilbert series

Purpose: Compute the truncated Hilbert series of K<X>/<G> up to a degree bound

Assume: - basering is a Letterplace ring.
- if you specify a different degree bound degbound,
degbound <= attrib(basering,uptodeg) holds.

Theory: Hilbert series of an algebra K<X>/<G> is sum_(i>=0) h_i t^i, where h_i is the K-dimension of the space of monomials of degree i, not contained in <G>. For finitely presented algebras Hilbert series NEED NOT be a rational function, though it happens often. Therefore in general there is no notion of a Hilbert polynomial.

Note: - If degbound is set, there will be a degree bound added. 0 means no
degree bound. Default: attrib(basering,uptodeg).
- n is the number of variables, which can be set to a different number.
Default: attrib(basering, lV).
- In the output intvec I, I[k] is the (k-1)-th coefficient of the Hilbert
series, i.e. h_(k-1) as above.

Example:

```
LIB "fpadim.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
setring R; // sets basering to Letterplace ring
ideal G = y*y,x*y*x; // G is a Groebner basis
lpHilbert(G); // procedure with default parameters
↦ 1,2,3,4,4,4
lpHilbert(G,3,2); // invokes procedure with degree bound 3 and (same) 2 variables
↦ 1,2,3,4
```

See also: Section 7.10.5 [ncHilb_lib], page 672.

7.10.1.6 teach_lpSickleDim

Procedure from library `fpadim.lib` (see Section 7.10.1 [fpadim_lib], page 639).

Usage: teach_lpSickleDim(G[,degbound,n]); G an ideal, degbound, n optional integers

Return: list

Purpose: Compute the K-dimension and the mistletoes of K<X>/<G>

Assume: - basering is a Letterplace ring.
- if you specify a different degree bound degbound,
degbound <= attrib(basering,uptodeg) holds.

Note: - If L is the list returned, then L[1] is an integer, the K-dimension,
L[2] is an ideal, the mistletoes.
- If degbound is set, there will be a degree bound added. 0 means no
degree bound. Default: attrib(basering,uptodeg).
- n is the number of variables, which can be set to a different number.
Default: attrib(basering, lV).
- If the K-dimension is known to be infinite, a degree bound is needed

Example:

```
LIB "fpadim.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
setring R; // sets basering to Letterplace ring
ideal G = x*x, y*y,x*y*x; // G is a monomial Groebner basis
teach_lpSickleDim(G); // invokes the procedure with ring parameters
↦ [1]:
↦    6
↦ [2]:
↦    _[1]=x*y
↦    _[2]=y*x*y
// the factor algebra is finite, so the degree bound, given
// by the Letterplace ring is not necessary
teach_lpSickleDim(G,0); // procedure without any degree bound
↦ [1]:
↦    6
↦ [2]:
↦    _[1]=x*y
↦    _[2]=y*x*y
```

7.10.2 fpalgebras_lib

Library: fpalgebras.lib

Purpose: Definitions of some finitely presented algebras and groups (Letterplace)

Authors: Karim Abou Zeid, karim.abou.zeid at rwth-aachen.de
Viktor Levandovskyy, viktor.levandovskyy at math.rwth-aachen.de
Grischa Studzinski, grischa.studzinski at rwth-aachen.de

Support: Project II.6 in the transregional collaborative research centre SFB-TRR 195 'Symbolic Tools in Mathematics and their Application' of the German DFG

Overview: Generation of various algebras, including group algebras of finitely presented groups in the Letterplace ring. FPA stands for finitely presented algebra.

Procedures: See also: Section 7.7 [LETTERPLACE], page 616; Section 7.10.1 [fpadim_lib], page 639; Section 7.10.3 [fpaprops_lib], page 659; Section 7.10.4 [freegb_lib], page 665.

7.10.2.1 operatorAlgebra

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: operatorAlgebra(a,d); a a string, d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - a gives the name of the algebra
- d gives the degreebound for the Letterplace ring

a must be one of the following:
integrodiff3
toeplitz
weyl1
usl2
usl2h
shift1inverse
exterior2
quadrowmm
shift1
weyl1inverse

This is a collection of common algebras.

Example:

```
LIB "fpalgebras.lib";
def R = operatorAlgebra("integrodiff3",5); setring R;
I; //relations of the algebra
↦ I[1]=-x*D+D*x-1
↦ I[2]=II*II-x*II+II*x
↦ I[3]=D*II-1
```

7.10.2.2 serreRelations

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: serreRelations(A,z); A an intmat, z an int

Return: ideal

Assume: basering has a letterplace ring structure and
A is a generalized Cartan matrix with integer entries

Purpose: compute the ideal of Serre's relations associated to A

Example:

```
LIB "fpalgebras.lib";
intmat A[3][3] =
2, -1, 0,
-1, 2, -3,
0, -1, 2; // G^1_2 Cartan matrix
ring r = 0,(f1,f2,f3),dp;
int uptodeg = 5;
def R = freeAlgebra(r, uptodeg);
setring R;
ideal I = serreRelations(A,1); I = simplify(I,1+2+8);
I;
↦ I[1]=f2*f2*f1-2*f2*f1*f2+f1*f2*f2
↦ I[2]=f3*f1-f1*f3
↦ I[3]=f2*f1*f1-2*f1*f2*f1+f1*f1*f2
↦ I[4]=f3*f3*f3*f3*f2-4*f3*f3*f3*f2*f3+6*f3*f3*f2*f3*f3-4*f3*f2*f3*f3*f3+f2\
   *f3*f3*f3*f3
↦ I[5]=f3*f2*f2-2*f2*f3*f2+f2*f2*f3
```

7.10.2.3 fullSerreRelations

Procedure from library fpalgebras.lib (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: fullSerreRelations(A,N,C,P,d); A an intmat, N,C,P ideals, d an int

Return: ring (and ideal)

Purpose: compute the inhomogeneous Serre's relations associated to A in given variable names

Assume: three ideals in the input are of the same sizes and contain merely variables which are interpreted as follows: N resp. P stand for negative resp. positive roots, C stand for Cartan elements. d is the degree bound for letterplace ring, which will be returned. The matrix A is a generalized Cartan matrix with integer entries The result is the ideal called 'fsRel' in the returned ring.

Example:

```
LIB "fpalgebras.lib";
intmat A[2][2] =
2, -1,
-1, 2; // A_2 = sl_3 Cartan matrix
ring r = 0,(f1,f2,h1,h2,e1,e2),dp;
ideal negroots = f1,f2; ideal cartans = h1,h2; ideal posroots = e1,e2;
int uptodeg = 5;
def RS = fullSerreRelations(A,negroots,cartans,posroots,uptodeg);
setring RS; fsRel;
↦ fsRel[1]=f2*f2*f1-2*f2*f1*f2+f1*f2*f2
↦ fsRel[2]=f2*f1*f1-2*f1*f2*f1+f1*f1*f2
↦ fsRel[3]=e2*e2*e1-2*e2*e1*e2+e1*e2*e2
↦ fsRel[4]=e2*e1*e1-2*e1*e2*e1+e1*e1*e2
↦ fsRel[5]=e1*f2-f2*e1
↦ fsRel[6]=e2*f1-f1*e2
↦ fsRel[7]=e1*f1-f1*e1-h1
↦ fsRel[8]=e2*f2-f2*e2-h2
↦ fsRel[9]=h2*h1-h1*h2
↦ fsRel[10]=e1*h1-h1*e1+2*e1
↦ fsRel[11]=h1*f1-f1*h1+2*f1
↦ fsRel[12]=e2*h1-h1*e2-e2
↦ fsRel[13]=h1*f2-f2*h1-f2
↦ fsRel[14]=e1*h2-h2*e1-e1
↦ fsRel[15]=h2*f1-f1*h2-f1
↦ fsRel[16]=e2*h2-h2*e2+2*e2
↦ fsRel[17]=h2*f2-f2*h2+2*f2
```

7.10.2.4 ademRelations

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: ademRelations(i,j); i,j int

Return: ring (and exports ideal)

Purpose: compute the ideal of Adem relations for i<2j in characteristic 0 the ideal is exported under the name AdemRel in the output ring

Example:

```
LIB "fpalgebras.lib";
def A = ademRelations(2,5);
setring A;
AdemRel;
↦ 6*s(7)*s(0)+s(6)*s(1)
```

7.10.2.5 baumslagSolitar

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: baumslagSolitar(m,n,d[,IsGroup]); n an integer, m an integer, d an integer, IsGroup an optional integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - in the group case: A = a^(-1), B = b^(-1)
- negative input is only allowed in the group case!
- d gives a degreebound and must be >m,n
- varying n and m produces a family of examples

Example:

```
LIB "fpalgebras.lib";
def R = baumslagSolitar(2,3,4); setring R;
I;
↦ I[1]=-b*a*a*a+a*a*b
↦ I[2]=a*A-1
↦ I[3]=b*B-1
↦ I[4]=a*A-A*a
↦ I[5]=b*B-B*b
```

7.10.2.6 baumslagGroup

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: baumslagGroup(m,n,d); m an integer, n an integer, d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - Baumslag group with the following presentation
< a, b | a^m = b^n = 1 >
-d gives the degreebound for the Letterplace ring
- varying n and m produces a family of examples

Example:

```
LIB "fpalgebras.lib";
def R = baumslagGroup(2,3,4); setring R;
I;
↦ I[1]=a*a-1
↦ I[2]=b*b*b-1
```

7.10.2.7 crystallographicGroupP1

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP1(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p1 group with the following presentation
< x, y | [x, y] = 1 >
-d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP1(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=X*x+1
↦ I[3]=x*X+1
↦ I[4]=y*Y+1
↦ I[5]=Y*y+1
```

7.10.2.8 crystallographicGroupPM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupPM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - pm group with the following presentation
< x, y, m | [x, y] = m^2 = 1, m^(-1)*x*m = x, m^(-1)*y*m = y^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupPM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=y*x+x*y+m*m
↦ I[3]=m*m+1
↦ I[4]=m*x*m+x
↦ I[5]=m*y*m+Y
↦ I[6]=x*X+1
↦ I[7]=X*x+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.9 crystallographicGroupPG

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupPG(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - pg group with the following presentation
< x, y, t | [x, y] = 1, t^2 = x, t^(-1)*y*t = y^(-1) >
- d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupPG(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=t*t+x
↦ I[3]=T*y*t+Y
↦ I[4]=X*x+1
↦ I[5]=x*X+1
↦ I[6]=Y*y+1
↦ I[7]=y*Y+1
↦ I[8]=t*T+1
↦ I[9]=T*t+1
```

7.10.2.10 crystallographicGroupP2MM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP2MM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p2mm group with the following presentation
< x, y, p, q | [x, y] = [p, q] = p^2 = q^2 = 1, p^(-1)*x*p = x, q^(-1)*x*q = x^(-1), p^(-1)*y*p = y^(-1), q^(-1)*y*q = y > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP2MM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=q*p+p*q+1
↦ I[3]=p*p+1
↦ I[4]=q*q+1
↦ I[5]=p*y*p+Y
↦ I[6]=p*x*p+x
↦ I[7]=q*y*q+y
↦ I[8]=q*x*q+X
↦ I[9]=X*x+1
↦ I[10]=x*X+1
↦ I[11]=Y*y+1
↦ I[12]=y*Y+1
↦ I[13]=y*x+x*y+p*p
↦ I[14]=y*x+x*y+q*q
↦ I[15]=p*p+q*q
```

7.10.2.11 crystallographicGroupP2

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP2(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p2 group with the following presentation
< x, y, m, t | [x, y] = t^2 = 1, m^2 = y, t^(-1)*x*t = x, m^(-1)*x*m = x^(-1), t^(-1)*y*t = y^(-1), t^(-1)*m*t = m^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP2(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=y*x+x*y+t*t
↦ I[3]=m*m+y
↦ I[4]=t*t+1
↦ I[5]=t*x*t+x
↦ I[6]=M*x*m+X
↦ I[7]=t*y*t+Y
↦ I[8]=t*m*t+M
↦ I[9]=X*x+1
↦ I[10]=x*X+1
↦ I[11]=Y*y+1
↦ I[12]=y*Y+1
↦ I[13]=m*M+1
↦ I[14]=M*m+1
```

7.10.2.12 crystallographicGroupP2GG

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP2GG(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p2gg group with the following presentation
< x, y, u, v | [x, y] = (u*v)^2 = 1, u^2 = x, v^2 = y, v^(-1)*x*v = x^(-1), u^(-1)*y*u = y^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP2GG(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=u*v*u*v+y*x+x*y
↦ I[3]=u*v*u*v+1
↦ I[4]=u*u+x
↦ I[5]=v*v+y
↦ I[6]=V*x*v+X
↦ I[7]=U*y*u+Y
↦ I[8]=X*x+1
↦ I[9]=x*X+1
↦ I[10]=Y*y+1
↦ I[11]=y*Y+1
↦ I[12]=u*U+1
↦ I[13]=U*u+1
↦ I[14]=v*V+1
↦ I[15]=V*v+1
```

7.10.2.13 crystallographicGroupCM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupCM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - cm group with the following presentation
< x, y, t | [x, y] = t^2 = 1, t^(-1)*x*t = x*y, t^(-1)*y*t = y^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupCM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=y*x+x*y+t*t
↦ I[3]=t*t+1
↦ I[4]=t*x*t+x*y
↦ I[5]=t*y*t+Y
↦ I[6]=X*x+1
↦ I[7]=x*X+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.14 crystallographicGroupC2MM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupC2MM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - c2mm group with the following presentation
< x, y, m, r | [x, y] = m^2 = r^2 = 1, m^(-1)*y*m = y^(-1), m^(-1)*x*m = x*y, r^(-1)*y*r = y^(-1), r^(-1)*x*r = x^(-1), m^(-1)*r*m = r^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupC2MM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=y*x+x*y+m*m
↦ I[3]=y*x+x*y+r*r
↦ I[4]=m*m+1
↦ I[5]=r*r+1
↦ I[6]=m*m+r*r
↦ I[7]=m*y*m+Y
↦ I[8]=m*x*m+x*y
↦ I[9]=r*y*r+Y
↦ I[10]=r*x*r+X
↦ I[11]=m*r*m+r
↦ I[12]=X*x+1
↦ I[13]=x*X+1
↦ I[14]=Y*y+1
↦ I[15]=y*Y+1
```

7.10.2.15 crystallographicGroupP4

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP4(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p4 group with the following presentation
< x, y, r | [x, y] = r^4 = 1, r^(-1)*x*r = x^(-1), r^(-1)*x*r = y > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP4(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r*r+y*x+x*y
↦ I[3]=r*r*r*r+1
↦ I[4]=r*r*r*x*r+X
↦ I[5]=r*r*r*x*r+y
↦ I[6]=X*x+1
↦ I[7]=x*X+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.16 crystallographicGroupP4MM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP4MM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p4mm group with the following presentation
< x, y, r, m | [x, y] = r^4 = m^2 = 1, r^(-1)*y*r = x^(-1), r^(-1)*x*r = y, m^(-1)*x*m = y, m^(-1)*r*m = r^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP4MM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r*r+y*x+x*y
↦ I[3]=r*r*r*r+1
↦ I[4]=r*r*r*x*r+X
↦ I[5]=r*r*r*x*r+y
↦ I[6]=X*x+1
↦ I[7]=x*X+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.17 crystallographicGroupP4GM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP4GM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p4gm group with the following presentation
< x, y, r, t | [x, y] = r^4 = t^2 = 1, r^(-1)*y*r = x^(-1), r^(-1)*x*r = y, t^(-1)*x*t = y, t^(-1)*r*t = x^(-1)*r^(-1)> - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP4GM(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r*r+y*x+x*y
↦ I[3]=r*r*r*r+1
↦ I[4]=y*x+x*y+t*t
↦ I[5]=t*t+1
↦ I[6]=r*r*r*r+t*t
↦ I[7]=r*r*r*y*r+X
↦ I[8]=r*r*r*x*r+y
↦ I[9]=X*r*r*r+t*r*t
↦ I[10]=X*x+1
↦ I[11]=x*X+1
↦ I[12]=Y*y+1
↦ I[13]=y*Y+1
```

7.10.2.18 crystallographicGroupP3

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP3(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p3 group with the following presentation
< x, y, r | [x, y] = r^3 = 1, r^(-1)*x*r = x^(-1)*y, r^(-1)*y*r = x^(-1)> - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP3(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r+y*x+x*y
↦ I[3]=r*r*r+1
↦ I[4]=r*r*x*r+X*y
↦ I[5]=r*r*y*r+X
↦ I[6]=X*x+1
↦ I[7]=x*X+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.19 crystallographicGroupP31M

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP31M(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p31m group with the following presentation
< x, y, r, t | [x, y] = r^2 = t^2 = (t*r)^3 = 1, r^(-1)*x*r = x, t^(-1)*y*t = y, t^(-1)*x*t = x^(-1)*y, r^(-1)*y*r = x*y^(-1) > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP31M(6); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=y*x+x*y+r*r
↦ I[3]=y*x+x*y+t*t
↦ I[4]=r*r+1
↦ I[5]=t*t+1
↦ I[6]=t*r*t*r*t*r+1
↦ I[7]=r*r+t*t
↦ I[8]=t*r*t*r*t*r+y*x+x*y
↦ I[9]=t*r*t*r*t*r+r*r
↦ I[10]=t*r*t*r*t*r+t*t
↦ I[11]=r*x*r+x
↦ I[12]=t*y*t+y
↦ I[13]=t*x*t+X*y
↦ I[14]=r*y*r+x*Y
↦ I[15]=X*x+1
↦ I[16]=x*X+1
↦ I[17]=Y*y+1
↦ I[18]=y*Y+1
```

7.10.2.20 crystallographicGroupP3M1

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP3M1(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p3m1 group with the following presentation
< x, y, r, m | [x, y] = r^3 = m^2 = 1, m^(-1)*r*m = r^2, r^(-1)*x*r = x^(-1)*y, r^(-1)*y*r = x^(-1), m^(-1)*x*m = x^(-1), m^(-1)*y*m = x^(-1)*y > - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP3M1(5); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r+y*x+x*y
↦ I[3]=y*x+x*y+m*m
↦ I[4]=r*r*r+1
↦ I[5]=m*m+1
↦ I[6]=r*r*r+m*m
↦ I[7]=m*r*m+r*r
↦ I[8]=r*r*x*r+X*y
↦ I[9]=r*r*y*r+X
↦ I[10]=m*x*m+X
↦ I[11]=m*y*m+X*y
↦ I[12]=X*x+1
↦ I[13]=x*X+1
↦ I[14]=Y*y+1
↦ I[15]=y*Y+1
```

7.10.2.21 crystallographicGroupP6

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP6(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p6 group with the following presentation
< x, y, r | [x, y] = r^6 = 1, r^(-1)*x*r = y, r^(-1)*y*r = x^(-1)*y> - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP6(7); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r*r*r*r+y*x+x*y
↦ I[3]=r*r*r*r*r*r+1
↦ I[4]=r*r*r*r*r*x*r+y
↦ I[5]=r*r*r*r*r*y*r+X*y
↦ I[6]=X*x+1
↦ I[7]=x*X+1
↦ I[8]=Y*y+1
↦ I[9]=y*Y+1
```

7.10.2.22 crystallographicGroupP6MM

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: crystallographicGroupP6MM(d); d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - p6mm group with the following presentation
< x, y, r, m | [x, y] = r^6 = m^2 = 1, r^(-1)*y*r = x^(-1)*y, r^(-1)*x*r = y, m^(-1)*x*m = x^(-1), m^(-1)*y*m = x^(-1)*y, m^(-1)*r*m = r^(-1)*y> - d gives the degreebound for the Letterplace ring

Example:

```
LIB "fpalgebras.lib";
def R = crystallographicGroupP6MM(7); setring R;
I;
↦ I[1]=y*x+x*y+1
↦ I[2]=r*r*r*r*r*r+y*x+x*y
↦ I[3]=r*r*r*r*r*r+1
↦ I[4]=y*x+x*y+m*m
↦ I[5]=r*r*r*r*r*r+m*m
↦ I[6]=m*m+1
↦ I[7]=m*x*m+X
↦ I[8]=m*y*m+X*y
↦ I[9]=r*r*r*r*r*x*r+y
↦ I[10]=r*r*r*r*r*y*r+X*y
↦ I[11]=r*r*r*r*r*y+m*r*m
↦ I[12]=X*x+1
↦ I[13]=x*X+1
↦ I[14]=Y*y+1
↦ I[15]=y*Y+1
```

7.10.2.23 dyckGroup1

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: dyckGroup1(n,d,P); n an integer, d an integer, P an intvec

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - The Dyck group with the following presentation
< x_1, x_2, ... , x_n | (x_1)^p1 = (x_2)^p2 = ... = (x_n)^pn = x_1 * x_2 * ... * x_n = 1 > - negative exponents are allowed
- representation in the form x_i^p_i - x_(i+1)^p_(i+1)
- d gives the degreebound for the Letterplace ring
- varying n and P produces a family of examples

Example:

```
LIB "fpalgebras.lib";
intvec P = 1,2,3;
def R = dyckGroup1(3,5,P); setring R;
I;
↦ I[1]=x(2)*x(2)+x(1)
↦ I[2]=x(3)*x(3)*x(3)+x(2)*x(2)
↦ I[3]=x(1)*x(2)*x(3)+x(3)*x(3)*x(3)
↦ I[4]=x(1)*x(2)*x(3)+1
```

7.10.2.24 dyckGroup2

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: dyckGroup2(n,d,P); n an integer, d an integer, P an intvec

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - The Dyck group with the following presentation
< x_1, x_2, ... , x_n | (x_1)^p1 = (x_2)^p2 = ... = (x_n)^pn = x_1 * x_2 * ... * x_n = 1 > - negative exponents are allowed
- representation in the form x_i^p_i - 1
- d gives the degreebound for the Letterplace ring
- varying n and P produces a family of examples

Example:

```
LIB "fpalgebras.lib";
intvec P = 1,2,3;
def R = dyckGroup2(3,5,P); setring R;
I;
↦ I[1]=x(1)+1
↦ I[2]=x(2)*x(2)+1
↦ I[3]=x(3)*x(3)*x(3)+1
↦ I[4]=x(1)*x(2)*x(3)+1
```

7.10.2.25 dyckGroup3

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: dyckGroup2(n,d,P); n an integer, d an integer, P an intvec

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - The Dyck group with the following presentation
< x_1, x_2, ... , x_n | (x_1)^p1 = (x_2)^p2 = ... = (x_n)^pn = x_1 * x_2 * ... * x_n = 1 > - only positive exponents are allowed
- no inverse generators needed
- d gives the degreebound for the Letterplace ring
- varying n and P produces a family of examples

Example:

```
LIB "fpalgebras.lib";
intvec P = 1,2,3;
def R = dyckGroup3(3,5,P); setring R;
I;
↦ I[1]=x(1)+1
↦ I[2]=x(2)*x(2)+1
↦ I[3]=x(3)*x(3)*x(3)+1
↦ I[4]=x(1)*x(2)*x(3)+1
```

7.10.2.26 fibonacciGroup

Procedure from library fpalgebras.lib (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: fibonacciGroup(m,d); m an integer, d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - The Fibonacci group F(2, m) with the following presentation < x_1, x_2, ... , x_m | x_i * x_(i + 1) = x_(i + 2) >
- d gives the degreebound for the Letterplace ring
- varying m produces a family of examples

Example:

```
LIB "fpalgebras.lib";
def R = fibonacciGroup(3,5); setring R;
I;
↦ I[1]=x(1)*x(2)+x(3)
↦ I[2]=x(1)*Y(1)+1
↦ I[3]=Y(1)*x(1)+1
↦ I[4]=x(2)*Y(2)+1
↦ I[5]=Y(2)*x(2)+1
↦ I[6]=x(3)*Y(3)+1
↦ I[7]=Y(3)*x(3)+1
```

7.10.2.27 tetrahedronGroup

Procedure from library fpalgebras.lib (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: tetrahedronGroup(g,d); g an integer, d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - g gives the number of the example (1 - 5)

- d gives the degreebound for the Letterplace ring
- varying g produces a family of examples

The examples are found in “Classification of the finite generalized tetrahedron groups” by Gerhard Rosenberger and Martin Scheer.
The 5 examples originate from Proposition 1.9 and describe finite generalized tetrahedron group in the Tsaranov-case, which are not equivalent to a presentation for an ordinary tetrahedron group.

Example:

```
LIB "fpalgebras.lib";
def R = tetrahedronGroup(3,5); setring R;
I;
↦ I[1]=x*x*x+1
↦ I[2]=y*y*y+1
↦ I[3]=z*z*z+1
↦ I[4]=x*y*x*y+1
↦ I[5]=x*z*x*z+1
↦ I[6]=y*z*y*z+1
```

7.10.2.28 triangularGroup

Procedure from library `fpalgebras.lib` (see Section 7.10.2 [fpalgebras_lib], page 645).

Usage: triangularGroup(g,d); g an integer, d an integer

Return: ring

Note: - the ring contains the ideal I, which contains the required relations - g gives the number of the example (1 - 14)
- d gives the degreebound for the Letterplace ring
- varying g produces a family of examples

The examples are found in
Classification of the finite generalized tetrahedron groups by Gerhard Rosenberger and Martin Scheer.
The 14 examples are denoted in theorem 2.12

Example:

```
LIB "fpalgebras.lib";
def R = triangularGroup(3,10); setring R;
I;
↦ I[1]=a*a*a+1
↦ I[2]=b*b*b+1
↦ I[3]=a*b*a*b*b*a*b*a*b*b+1
```

7.10.3 fpaprops_lib

Library: fpaprops.lib

Purpose: Algorithmic ring-theoretic properties of finitely presented algebras (Letterplace)

Authors: Karim Abou Zeid, karim.abou.zeid at rwth-aachen.de

Support: Project II.6 in the transregional collaborative research centre SFB-TRR 195 ’Symbolic Tools in Mathematics and their Application’ of the German DFG

Overview: In this library, algorithms for computing various ring-theoretic properties of finitely presented algebras are implemented.
Applicability: Letterplace rings.

References:
Huishi Li: Groebner bases in ring theory. World Scientific, 2010.

Procedures: See also: Section 7.10.1 [fpadim_lib], page 639; Section 7.10.4 [freegb_lib], page 665.

7.10.3.1 lpNoetherian

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpNoetherian(G); G an ideal in a Letterplace ring

Return: int
0 not Noetherian
1 left Noetherian
2 right Noetherian
3 Noetherian
4 weak Noetherian

Purpose: Check whether the monomial algebra A/<LM(G)> is (left/right) noetherian

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Theory: lpNoetherian works with the monomial algebra A/<LM(G)>. If it gives an affirmative answer for one of the properties, then it holds for both A/<LM(G)> and A/<G>. However, a negative answer applies only to A/<LM(G)> and not necessarily to A/<G>.

Note: Weak Noetherian means that two-sided ideals in A/<LM(G)> satisfy the acc (ascending chain condition).

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5);
setring R;
ideal G = x*x, y*x; // K<x,y>/<xx,yx> is right noetherian
lpNoetherian(G);
↦ 2
```

7.10.3.2 lpIsSemiPrime

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpIsSemiPrime(G); G an ideal in a Letterplace ring

Return: boolean

Purpose: Check whether A/<LM(G)> is semi-prime ring,
alternatively whether <LM(G)> is a semi-prime ideal in A.

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Theory: A (two-sided) ideal I in the ring A is semi-prime, if for any a in A one has aAa subseteq I implies a in I.

Note: lpIsSemiPrime works with the monomial algebra A/<LM(G)>. A positive answer holds for both A/<LM(G)> and A/<G>, while a negative answer applies only to A/<LM(G)> and not necessarily to A/<G>.

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x1,x2),dp;
def R = freeAlgebra(r, 5);
setring R;
ideal G = x1*x2, x2*x1; // K<x1,x2>/<x1*x2,x2*x1> is semi prime
lpIsSemiPrime(G);
↦ 1
```

7.10.3.3 lpIsPrime

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpIsPrime(G); G an ideal in a Letterplace ring

Return: boolean

Purpose: Check whether A/<LM(G)> is prime ring,
alternatively whether <LM(G)> is a prime ideal in A.

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Theory: A (two-sided) ideal I in the ring A is prime, if for any a,b in A one has aAb subseteq I implies a in I or b in I.

Note: lpIsPrime works with the monomial algebra A/<LM(G)>.
A positive answer holds for both A/<LM(G)> and A/<G>, while a negative answer applies only to A/<LM(G)> and not necessarily to A/<G>.

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5);
setring R;
ideal G = x*x, y*y; // K<x,y>/<xx,yy> is prime
lpIsPrime(G);
↦ 1
```

7.10.3.4 lpGkDim

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpGkDim(G); G an ideal in a letterplace ring

Return: int

Purpose: Determines the Gelfand Kirillov dimension of A/<G>
-1 means positive infinite

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Note: Alias for dim(G)

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r, 5);
ideal I = z; // infinite GK dimension (-1)
lpGkDim(I);
↦ WARNING: ‘lpGkDim‘ is deprecated, you can use ‘dim‘ instead.
↦ -1
I = x,y,z; I = std(I); // GK dimension 0
lpGkDim(I);
↦ WARNING: ‘lpGkDim‘ is deprecated, you can use ‘dim‘ instead.
↦ 0
I = x*y, x*z, z*y, z*z; I = std(I); // GK dimension 2
lpGkDim(I);
↦ WARNING: ‘lpGkDim‘ is deprecated, you can use ‘dim‘ instead.
↦ 2
ideal G = y*x - x*y, z*x - x*z, z*y - y*z; G = std(G);
G;
↦ G[1]=z*y-y*z
↦ G[2]=z*x-x*z
↦ G[3]=y*x-x*y
lpGkDim(G); // GK dimension 3
↦ WARNING: ‘lpGkDim‘ is deprecated, you can use ‘dim‘ instead.
↦ 3
```

7.10.3.5 teach_lpGkDim

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: teach_lpGkDim(G); G an ideal in a letterplace ring

Return: int

Purpose: Determines the Gelfand Kirillov dimension of A/<G>
-1 means positive infinite

Assume: - basering is a Letterplace ring
- G is a Groebner basis

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y,z),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
R;
↦ // coefficients: QQ
↦ // number of vars : 15
↦ //        block   1 : ordering dp
↦ //                  : names    x y z x y z x y z x y z x y z
↦ //        block   2 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
setring R; // sets basering to Letterplace ring
ideal I = z;//an example of infinite GK dimension
teach_lpGkDim(I);
↦ -1
I = x,y,z; // gkDim = 0
```

```
teach_lpGkDim(I);
↦ 0
I = x*y, x*z, z*y, z*z;//gkDim = 2
teach_lpGkDim(I);
↦ 2
ideal G = y*x - x*y, z*x - x*z, z*y - y*z; G = std(G);
G;
↦ G[1]=z*y-y*z
↦ G[2]=z*x-x*z
↦ G[3]=y*x-x*y
teach_lpGkDim(G); // 3, as expected for K[x,y,z]
↦ 3
```

7.10.3.6 lpGlDimBound

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpGlDimBound(I); I an ideal

Return: int, an upper bound for the global dimension, -1 means infinity

Purpose: computing an upper bound for the global dimension

Assume: - basering is a Letterplace ring, G is a reduced Groebner Basis

Note: if I = LM(I), then the global dimension is equal the Gelfand Kirillov dimension if it is finite
Global dimension should be 0 for A/G = K and 1 for A/G = K<x1...xn>

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y),dp;
def R = freeAlgebra(r, 5); // constructs a Letterplace ring
setring R; // sets basering to Letterplace ring
ideal G = x*x, y*y,x*y*x; // it is a monomial Groebner basis
lpGlDimBound(G);
↦ 0
ideal H = y*x - x*y; H = std(H); // H is a Groebner basis
lpGlDimBound(H); // gl dim of K[x,y] is 2, as expected
↦ 2
```

7.10.3.7 lpSubstitute

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpSubstitute(f,s1,s2[,G]); f poly, s1 list (ideal) of variables to replace, s2 list (ideal) of polynomials to replace with, G optional ideal to reduce with.

Return: poly, the substituted polynomial

Assume: - basering is a Letterplace ring
- s1 contains a subset of the set of variables
- s2 and s1 are of the same size
- G is a Groebner basis,
- the current ring has a sufficient degbound (which also can be calculated with lpCalcSubstDegBound())

Note: the procedure implements the image of a polynomial f
under an endomorphism of a free algebra, defined by s1 and s2: variables, not present in s1, are left unchanged;
variable s1[k] is mapped to a polynomial s2[k].
- An optional ideal G extends the endomorphism as above to the morphism into the factor algebra K<X>/G.

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y,z),dp;
def R = freeAlgebra(r, 4);
setring R;
ideal G = x*y; // optional
poly f = 3*x*x+y*x;
ideal s1 = x, y;
ideal s2 = y*z*z, x; // i.e. x --> yzz and y --> x
// the substitution probably needs a higher degbound
int minDegBound = lpCalcSubstDegBound(f,s1,s2);
minDegBound; // thus the bound needs to be increased
↦ 9
setring r; // back to original r
def R1 = freeAlgebra(r, minDegBound);
setring R1;
lpSubstitute(imap(R,f), imap(R,s1), imap(R,s2));
↦ 3*y*z*z*y*z*z+x*y*z*z
// the last parameter is optional; above it was G=<xy>
// the output will be reduced with respect to G
lpSubstitute(imap(R,f), imap(R,s1), imap(R,s2), imap(R,G));
↦ 3*y*z*z*y*z*z
```

7.10.3.8 lpCalcSubstDegBound

Procedure from library `fpaprops.lib` (see Section 7.10.3 [fpaprops_lib], page 659).

Usage: lpCalcSubstDegBound(I,s1,s2); I ideal of polynomials, s1 ideal of variables to replace, s2 ideal of polynomials to replace with

Return: int, the min degbound required to perform all of the substitutions

Assume: - basering is a Letterplace ring

Note: convenience method

Example:

```
LIB "fpaprops.lib";
ring r = 0,(x,y,z),dp;
def R = freeAlgebra(r, 4);
setring R;
ideal I = 3*x*x+y*x, x*y*x - z;
ideal s1 = x, y; // z --> z
ideal s2 = y*z*z, x; // i.e. x --> yzz and y --> x
// the substitution probably needs a higher degbound
lpCalcSubstDegBound(I,s1,s2);
↦ 10
lpCalcSubstDegBound(I[1],s1,s2);
↦ 9
```

7.10.4 freegb_lib

Library: freegb.lib

Purpose: Two-sided Groebner bases in free algebras and tools via Letterplace approach

Authors: Viktor Levandovskyy, viktor.levandovskyy at math.rwth-aachen.de
Karim Abou Zeid, karim.abou.zeid at rwth-aachen.de
Grischa Studzinski, grischa.studzinski at math.rwth-aachen.de

Overview: For the theory, see chapter 'Letterplace' in the Singular Manual.

This library provides access to kernel functions and also contains legacy code (partially as static procedures) for compatibility reasons.

Support: Joint projects LE 2697/2-1 and KR 1907/3-1 of the Priority Programme SPP 1489: 'Algorithmische und Experimentelle Methoden in Algebra, Geometrie und Zahlentheorie' of the German DFG and Project II.6 of the transregional collaborative research centre SFB-TRR 195 'Symbolic Tools in Mathematics and their Application' of the German DFG

Procedures: See also: Section 7.7 [LETTERPLACE], page 616; Section 7.10.1 [fpadim_lib], page 639; Section 7.10.2 [fpalgebras_lib], page 645; Section 7.10.3 [fpaprops_lib], page 659.

7.10.4.1 isFreeAlgebra

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: isFreeAlgebra(r); r a ring

Return: boolean

Purpose: check whether R is a letterplace ring (free algebra)

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
isFreeAlgebra(r);
↦ 0
ring R = freeAlgebra(r, 7);
isFreeAlgebra(R);
↦ 1
```

7.10.4.2 lpDegBound

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: lpDegBound(R); R a letterplace ring

Return: int

Purpose: returns the degree bound of the letterplace ring

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
def R = freeAlgebra(r, 7);
lpDegBound(R);
↦ 7
```

7.10.4.3 lpVarBlockSize

Procedure from library **freegb.lib** (see Section 7.10.4 [freegb_lib], page 665).

Usage: lpVarBlockSize(R); R a letterplace ring

Return: int

Purpose: returns the variable block size of the letterplace ring, that is the number of variables of the original ring.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r, 7);
lpVarBlockSize(R);
↦ 3
```

7.10.4.4 lpNcgenCount

Procedure from library **freegb.lib** (see Section 7.10.4 [freegb_lib], page 665).

Usage: lpNcgenCount(R); R a letterplace ring

Return: int

Purpose: returns the number of ncgen variables in the letterplace ring.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
ring R = freeAlgebra(r, 7, 3);
lpNcgenCount(R); // should be 3
↦ 3
```

7.10.4.5 lpDivision

Procedure from library **freegb.lib** (see Section 7.10.4 [freegb_lib], page 665).

Usage: lpDivision(p,G); poly p, ideal G

Purpose: compute a two-sided division with remainder of p wrt G; two-sided noncommutative analogue of the procedure division

Assume: G = {g1,...,gN} is a Groebner basis, the original ring of the Letterplace ring has the name 'r' and no variable is called 'tag_i' for i in 1...N

Return: list L

Note: - L[1] is NF(p,I)
- L[2] is the list of expressions [i,l_(ij),r_(ij)] with \sum_(i,j) l_(ij) g_i r_(ij) = p - NF(p,I)
- procedure lpGBPres2Poly, applied to L, reconstructs p

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),dp;
ring R = freeAlgebra(r, 4);
ideal I = x*x + y*y - 1; // 2D sphere
ideal J = twostd(I); // compute a two-sided Groebner basis
```

```
J; // it is finite and nice
↦ J[1]=x*x+y*y-1
↦ J[2]=y*y*x-x*y*y
poly h = x*x*y-y*x*x+x*y;
list L = lpDivision(h,J); L; // what means that the NF of h wrt J is x*y
↦ [1]:
↦    x*y
↦ [2]:
↦    [1]:
↦       [1]:
↦          1
↦       [2]:
↦ 1
↦       [3]:
↦          y
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          -y
↦       [3]:
↦          1
h - lpNF(h,J); // and this poly has the following two-sided Groebner presentation:
↦ -y*x*x+x*x*y
-y*J[1] + J[1]*y;
↦ -y*x*x+x*x*y
lpGBPres2Poly(L,J); // reconstructs the above automatically
↦ -y*x*x+x*x*y+x*y
```

7.10.4.6 lpGBPres2Poly

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: lpGBPres2Poly(p,G); poly p, ideal G

Assume: L is a valid Groebner presentation like the result of lpDivision

Return: poly

Note: assembles p = \sum_(i,j) l_(ij) g_i r_(ij) + NF(p,I) = \sum_(i) L[2][i][2] I[L[2][i][1]] L[2][i][3] + L[1]

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),dp;
ring R = freeAlgebra(r, 4);
ideal I = x*x + y*y - 1; // 2D sphere
ideal J = twostd(I); // compute a two-sided Groebner basis
J; // it is finite and nice
↦ J[1]=x*x+y*y-1
↦ J[2]=y*y*x-x*y*y
poly h = x*x*y-y*x*x+x*y;
list L = lpDivision(h,J);
L[1]; // what means that the normal form (or the remainder) of h wrt J is x*y
↦ x*y
```

```
lpGBPres2Poly(L,J); // we see, that it is equal to h from above
↦ -y*x*x+x*x*y+x*y
```

7.10.4.7 isOrderingShiftInvariant

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: isOrderingShiftInvariant(b); b an integer interpreted as a boolean

Return: int

Note: Tests whether the ordering of the current ring is shift invariant, which is the case, when LM(p) > LM(p') for all p and p' where p' is p shifted by any number of places.

If withHoles != 0 even Letterplace polynomials with holes (eg. x(1)*y(4)) are considered.

Assume: - basering is a Letterplace ring.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
def R = freeAlgebra(r, 5);
setring R;
isOrderingShiftInvariant(0);// should be 1
↦ 1
ring r2 = 0,(x,y,z),dp;
def R2 = freeAlgebra(r2, 5);
list RL = ringlist(R2);
RL[3][1][1] = "wp";
intvec weights = 1,1,1,1,1,1,1,2,3,1,1,1,1,1,1;
RL[3][1][2] = weights;
attrib(RL,"isLetterplaceRing",3);
attrib(RL,"maxExp",1);
def Rw = setLetterplaceAttributes(ring(RL),5,3);
setring Rw;
/* printlevel = voice + 1; */
isOrderingShiftInvariant(0);
↦ 0
isOrderingShiftInvariant(1);
↦ 0
```

7.10.4.8 makeLetterplaceRing

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: makeLetterplaceRing(d [,h]); d an integer, h an optional integer (deprecated, use freeAlgebra instead)

Return: ring

Purpose: creates a ring with the ordering, used in letterplace computations

Note: h = -1 (default) : the ordering of the current ring will be used h = 0 : Dp ordering will be used
h = 2 : weights 1 used for all the variables, a tie breaker is a list of block of original ring
h = 1 : the pure homogeneous letterplace block ordering (applicable in the situation of homogeneous input ideals) will be used.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),Dp;
def A = makeLetterplaceRing(2); // same as  makeLetterplaceRing(2,0)
setring A;  A;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering Dp
↦ //                  : names    x y z x y z
↦ //        block   2 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
lpVarBlockSize(A);
↦ 3
lpDegBound(A);  // degree bound
↦ 2
setring r; def B = makeLetterplaceRing(2,1); // to compare:
setring B;  B;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering Dp
↦ //                  : names    x y z
↦ //        block   2 : ordering Dp
↦ //                  : names    x y z
↦ //        block   3 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
lpVarBlockSize(B);
↦ 3
lpDegBound(B);  // degree bound
↦ 2
setring r; def C = makeLetterplaceRing(2,2); // to compare:
setring C;  C;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering a
↦ //                  : names    x y z x y z
↦ //                  : weights  1 1 1 1 1 1
↦ //        block   2 : ordering Dp
↦ //                  : names    x y z
↦ //        block   3 : ordering Dp
↦ //                  : names    x y z
↦ //        block   4 : ordering C
↦ // letterplace ring (block size 3, ncgen count 0)
lpDegBound(C);
↦ 2
lpDegBound(C);  // degree bound
↦ 2
```

7.10.4.9 letplaceGBasis

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: letplaceGBasis(I); I an ideal/module

Return: ideal/module

Assume: basering is a Letterplace ring, input consists of Letterplace polynomials

Purpose: compute the two-sided Groebner basis of I via Letterplace algorithm (legacy routine)

Note: the degree bound for this computation is read off the letterplace structure of basering

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),Dp;
int degree_bound = 5;
def R = freeAlgebra(r, 5);
setring R;
ideal I = -x*y-7*y*y+3*x*x, x*y*x-y*x*y;
ideal J = letplaceGBasis(I);
J;
↦ J[1]=3*x*x-x*y-7*y*y
↦ J[2]=22*x*y*y-3*y*x*y-21*y*y*x+7*y*y*y
↦ J[3]=3*x*y*x-22*x*y*y+21*y*y*x-7*y*y*y
↦ J[4]=22803*y*y*y*x+19307*y*y*y*y
↦ J[5]=1933*y*y*x*y+2751*y*y*y*x+161*y*y*y*y
↦ J[6]=y*y*y*y*y
```

7.10.4.10 lieBracket

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: lieBracket(a,b[,N]); a,b letterplace polynomials, N an optional integer

Return: poly

Assume: basering has a letterplace ring structure

Purpose: compute the Lie bracket [a,b] = ab - ba between letterplace polynomials

Note: if N>1 is specified, then the left normed bracket [a,[...[a,b]]]] is computed.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),dp;
ring R = freeAlgebra(r, 4);
poly a = x*y; poly b = y;
lieBracket(a,b);
↦ -y*x*y+x*y*y
lieBracket(x,y,2);
↦ y*x*x-2*x*y*x+x*x*y
```

7.10.4.11 setLetterplaceAttributes

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: setLetterplaceAttributes(R, d, b); R a ring, b,d integers

Return: ring with special attributes set

Purpose: sets attributes for a letterplace ring:
'isLetterplaceRing' = 'lV' = b, 'uptodeg' = d, where 'uptodeg' stands for the degree bound,
'lV' for the number of variables in the block 0.

Note: Activate the resulting ring by using `setring`

Example:

```
LIB "freegb.lib";
ring r = 0,(x(1),y(1),x(2),y(2),x(3),y(3),x(4),y(4)),dp;
def R = setLetterplaceAttributes(r, 4, 2); setring R;
lpVarBlockSize(R);
↦ 2
lieBracket(x(1),y(1),2);
↦ y(1)*x(2)*x(3)-2*x(1)*y(2)*x(3)+x(1)*x(2)*y(3)
```

7.10.4.12 testLift

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: testLift(M,T); module M, matrix T

Return: module

Purpose: assembles the result of the lift procedure

Assume: T is the lift matrix of a submodule of M

Note: the inverse of the lift procedure

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),(c,Dp);
ring R = freeAlgebra(r, 7, 2);
ideal I = std(x*y*x + 1);
print(matrix(I)); // finite two-sided Groebner basis
↦ x*y-y*x,y*x*x+1
ideal SI = x*I[1]*y + y*x*I[2], I[1]*y*x + I[2]*y;
matrix T = lift(I, SI); // T is the lifting matrix of SI wrt I
print(T); //
↦ y*ncgen(1)*x*x+x*ncgen(1)*y,y*x*ncgen(1)+y*ncgen(1)*x+ncgen(1)*y*x,
↦ y*ncgen(2)*x,               y*ncgen(2)
print(matrix(SI)); // the original generators of SI as a matrix
↦ y*x*y*x*x+x*x*y*y-x*y*x*y+y*x,x*y*y*x+y*x*x*y-y*x*y*x+y
print(matrix(testLift(I,T))); // and the result of testLift
↦ y*x*y*x*x+x*x*y*y-x*y*x*y+y*x,x*y*y*x+y*x*x*y-y*x*y*x+y
```

7.10.4.13 testSyz

Procedure from library `freegb.lib` (see Section 7.10.4 [freegb_lib], page 665).

Usage: testSyz(M,S); module M, S

Return: module

Purpose: tests the result of the syz procedure

Assume: S is the syzygy bimodule of M

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),(c,Dp);
ring R = freeAlgebra(r, 7, 2);
```

```
ideal I = twostd(x*y*x + 1);
print(matrix(I));
↦ x*y-y*x,y*x*x+1
module S = syz(I);
print(S);
↦ ncgen(1)*x*x,S[1,2],S[1,3],S[1,4],S[1,5],
↦ S[2,1],      S[2,2],S[2,3],S[2,4],S[2,5]
testSyz(I,S); // returns zero
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
↦ _[4]=0
↦ _[5]=0
```

7.10.5 ncHilb_lib

Library: ncHilb.lib

Purpose: Computation of graded and multi-graded Hilbert series of non-commutative algebras (Letterplace).

Author: Sharwan K. Tiwari shrawant@gmail.com
Roberto La Scala
Viktor Levandovskyy (adaptation to the new Letterplace release)

References:

La Scala R.: Monomial right ideals and the Hilbert series of non-commutative modules, Journal of Symbolic Computation (2016).

La Scala R., Tiwari Sharwan K.: Multigraded Hilbert Series of noncommutative modules, https://arxiv.org/abs/1705.01083.

Procedures:

7.10.5.1 nchilb

Procedure from library `ncHilb.lib` (see Section 7.10.5 [ncHilb_lib], page 672).

Usage: nchilb(I, d[, L]), list I, int d, optional list L

Purpose: compute Hilbert series of a non-commutative algebra

Assume:

Note: d is an integer for the degree bound (maximal total degree of polynomials of the generating set of the input ideal),
#[]=1, computation for non-finitely generated regular ideals, #[]=2, computation of multi-graded Hilbert series,
#[]=tdeg, for obtaining the truncated Hilbert series up to the total degree tdeg-1 (tdeg should be > 2), and #[]=string(p), to print the details about the orbit and system of equations. Let the orbit is O_I = {T_{w_1}(I),...,T_{w_r}(I)} ($w_i\in W$), where we assume that if T_{w_i}(I)=T_{w_i'}(I)$ for some $w'_i\in W$, then $deg(w_i)\leq deg(w'_i)$.
Then, it prints words description of orbit: w_1,...,w_r. It also prints the maximal degree and the cardinality of \sum_j R(w_i, b_j) corresponding to each w_i, where {b_j} is a basis of I.
Moreover, it also prints the linear system (for the information about adjacency matrix) and its solving time.

Note : A Groebner basis of two-sided ideal of the input should be given in a special form. This form is a list of modules, where each generator of every module represents a monomial times a coefficient in the free associative algebra. The first entry, in each generator, represents a coefficient and every next entry is a variable.

Ex: module p1=[1,y,z],[-1,z,y], represents the poly y*z-z*y; module p2=[1,x,z,x],[-1,z,x,z], represents the poly x*z*x-z*x*z for more details about the input, see examples.

Example:

```
LIB "ncHilb.lib";
ring r=0,(X,Y,Z),dp;
module p1 =[1,Y,Z];             //represents the poly Y*Z
module p2 =[1,Y,Z,X];          //represents the poly Y*Z*X
module p3 =[1,Y,Z,Z,X,Z];
module p4 =[1,Y,Z,Z,Z,X,Z];
module p5 =[1,Y,Z,Z,Z,Z,X,Z];
module p6 =[1,Y,Z,Z,Z,Z,Z,X,Z];
module p7 =[1,Y,Z,Z,Z,Z,Z,Z,X,Z];
module p8 =[1,Y,Z,Z,Z,Z,Z,Z,Z,X,Z];
list l1=list(p1,p2,p3,p4,p5,p6,p7,p8);
nchilb(l1,10);
↦
↦ maximal length of words = 2
↦
↦ length of the Orbit = 3
↦
↦
↦ Hilbert series:
↦ 1/(t2-3t+1)
ring r2=0,(x,y,z),dp;
module p1=[1,y,z],[-1,z,y];               //y*z-z*y
module p2=[1,x,z,x],[-1,z,x,z];           // x*z*x-z*x*z
module p3=[1,x,z,z,x,z],[-1,z,x,z,z,x];   // x*z^2*x*z-z*x*z^2*x
module p4=[1,x,z,z,z,x,z],[-1,z,x,z,z,x,x]; // x*z^3*x*z-z*x*z^2*x^2
list l2=list(p1,p2,p3,p4);
nchilb(l2,6,1); //third argument '1' is for non-finitely generated case
↦
↦ maximal length of words = 3
↦
↦ length of the Orbit = 7
↦
↦
↦ Hilbert series:
↦ (t3+t2+1)/(2t5-2t4-t3+2t2-3t+1)
ring r3=0,(a,b),dp;
module p1=[1,a,a,a];
module p2=[1,a,b,b];
module p3=[1,a,a,b];
list l3=list(p1,p2,p3);
nchilb(l3,5,2);//third argument '2' is to compute multi-graded HS
↦
↦ maximal length of words = 3
↦
↦ length of the Orbit = 5
```

```
↦
↦
↦ Hilbert series:
↦ (t1^2+t1+1)/(t1*t2^2-t1*t2-t2+1)
ring r4=0,(x,y,z),dp;
module p1=[1,x,z,y,z,x,z];
module p2=[1,x,z,x];
module p3=[1,x,z,y,z,z,x,z];
module p4=[1,y,z];
module p5=[1,x,z,z,x,z];
list l4=list(p1,p2,p3,p4,p5);
nchilb(l4,7,"p"); //third argument "p" is to print the details
↦
↦ maximal length of words = 3
↦
↦ length of the Orbit = 6
↦ words description of the Orbit:
↦ 1    x    y    x*z    y*z    x*z*z
↦
↦ maximal degree,  #(sum_j R(w,w_j))
↦ NULL
↦ 6,  4
↦ 1,  1
↦ 5,  4
↦ 0,  1
↦ 2,  1
↦
↦ linear system:
↦ H(1) = (t)*H(2) + (t)*H(3) + (t)*H(1) +  1
↦ H(2) = (t)*H(2) + (t)*H(3) + (t)*H(4) +  1
↦ H(3) = (t)*H(2) + (t)*H(3) + (t)*H(5) +  1
↦ H(4) = (t)*H(5) + (t)*H(3) + (t)*H(6) +  1
↦ H(5) = (t)*H(5) + (t)*H(5) + (t)*H(5) +  0
↦ H(6) = (t)*H(3) + (t)*H(3) + (t)*H(1) +  1
↦ where H(1) represents the series corresp. to input ideal
↦ and i^th summand in the rhs of an eqn. is according
↦ to the right colon map corresp. to the i^th variable
↦
↦
↦ Hilbert series:
↦ (t3+t2+1)/(2t5-2t4-t3+2t2-3t+1)
// of the orbit and system
```

7.10.5.2 rcolon

Procedure from library `ncHilb.lib` (see Section 7.10.5 [ncHilb_lib], page 672).

Usage: rcolon(list of relations, a monomial, an integer);
L is a list of modules (each module represents a monomial), w is a monomail
d is an integer for the degree bound (maximal total degree of monomials of the generating set of the input monomial ideal),

Note : A two-sided monomial ideal and a monomial w for the input should be given in a special form. This form is a list of modules, where the generator of every module represents a monomial times a coefficient in the free associative algebra. The first entry, in each generator, represents a coefficient, that is 1, and every next entry is a variable.

Ex: module p1=[1,y,z], represents the monomial y*z;
module p2=[1,x,z,x], represents the monomial x*z*x
for more details about the input, see examples.

Example:

```
LIB "ncHilb.lib";
ring r=0,(X,Y,Z),dp;
module w =[1,Y];
module p1 =[1,Y,Z];
module p2 =[1,Y,Z,X];
module p3 =[1,Y,Z,Z,X,Z];
module p4 =[1,Y,Z,Z,Z,X,Z];
module p5 =[1,Y,Z,Z,Z,Z,X,Z];
module p6 =[1,Y,Z,Z,Z,Z,Z,X,Z];
module p7 =[1,Y,Z,Z,Z,Z,Z,Z,X,Z];
module p8 =[1,Y,Z,Z,Z,Z,Z,Z,Z,X,Z];
list l1=list(p1,p2,p3,p4,p5,p6,p7,p8);
rcolon(l1,w,10);
↦ J[1]=Z
↦ + generators of the given ideal;
```

7.10.6 ncrat_lib

Status: experimental

Library: ncrat.lib

Purpose: Framework for working with non-commutative rational functions

Author: Ricardo Schnur, email: ricardo.schnur@math.uni-sb.de

Support: This project has been funded by the SFB-TRR 195
'Symbolic Tools in Mathematics and their Application'.

Overview: This library provides a framework for working with
non-commutative rational functions (or rather, expressions) and their linearized representations

References:

T. Mai: On the analytic theory of non-commutative distributions in free probability. Universitaet des Saarlandes, Dissertation, 2017

Note: an almost self-explaining introduction to the possibilities of the framework can be achieved by running the example for the procedure ncrepGetRegularMinimal.

Procedures:

7.10.6.1 ncInit

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncInit(vars);
list vars containing strings

Return: datatypes ncrat and ncrep (and token, tokenstream,
but they are not meant for users),
sets ring as 'NCRING' with nc variables from list l

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
NCRING;
↦ // coefficients: QQ[I]/(I^2+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

7.10.6.2 ncVarsGet

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncVarsGet();

Returns: nc variables that are in use

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncVarsGet();
↦ x,y,z
```

7.10.6.3 ncVarsAdd

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncVarsAdd(vars);
list vars contains variables

Returns: sets list elements as nc variables

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncVarsGet();
↦ x,y,z
ncVarsAdd(list("a", "b", "c"));
↦ // ** killing the basering for level 0
ncVarsGet();
↦ x,y,z,a,b,c
```

7.10.6.4 ncratDefine

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat f = ncratDefine(s, l);
string s contains kind, list l contains expressions

Return: ncrat with kind s and expressions l

Note: assignment operator '=' for ncrat is overloaded
with this procedure, hence
ncrat f = s, l;
yields the same result as
ncrat f = ncratDefine(s, l);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
number n = 5;
ncrat f = ncratDefine("Const", list(n));
typeof(f);
↦ ncrat
f.kind;
↦ Const
f.expr;
↦ [1]:
↦ 5
f;
↦ 5
↦
ncrat g = "Const", list(n);
g;
↦ 5
↦
```

7.10.6.5 ncratAdd

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat h = ncratAdd(f, g);
f, g both of type ncrat

Return: h = f + g

Note: operator '+' for ncrat is overloaded
with this procedure, hence
ncrat h = f + g;
yields the same result as
ncrat h = ncratAdd(f, g);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
print(f);
↦ 2*x*y
ncrat g = ncratFromString("z");
print(g);
↦ z
ncrat h1, h2;
h1 = ncratAdd(f, g);
print(h1);
↦ 2*x*y+z
h2 = f + g;
```

```
print(h2);
↦ 2*x*y+z
```

7.10.6.6 ncratSubstract

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat h = ncratSubstract(f, g);
f, g both of type ncrat

Return: h = f - g

Note: operator '-' for ncrat is overloaded
with this procedure, hence
ncrat h = f - g;
yields the same result as
ncrat h = ncratSubstract(f, g);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
print(f);
↦ 2*x*y
ncrat g = ncratFromString("z");
print(g);
↦ z
ncrat h1, h2;
h1 = ncratSubstract(f, g);
print(h1);
↦ 2*x*y-z
h2 = f - g;
print(h2);
↦ 2*x*y-z
```

7.10.6.7 ncratMultiply

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat h = ncratMultiply(f, g);
f, g both of type ncrat

Return: h = f * g

Note: operator '*' for ncrat is overloaded
with this procedure, hence
ncrat h = f * g;
yields the same result as
ncrat h = ncratMultiply(f, g);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
print(f);
↦ 2*x*y
```

```
ncrat g = ncratFromString("z");
print(g);
↦ z
ncrat h1, h2;
h1 = ncratMultiply(f, g);
print(h1);
↦ 2*x*y*z
h2 = f * g;
print(h2);
↦ 2*x*y*z
```

7.10.6.8 ncratInvert

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat h = ncratInvert(f);
f of type ncrat

Return: h = inv(f)

Note: ncrat h = f^-1;
yields the same result as
ncrat h = ncratInvert(f);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
print(f);
↦ 2*x*y
ncrat h1, h2;
h1 = ncratInvert(f);
print(h1);
↦ inv(2*x*y)
h2 = f ^ -1;
print(h2);
↦ inv(2*x*y)
```

7.10.6.9 ncratSPrint

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: string s = ncratSPrint(f);
f of type ncrat

Return: prints f to string

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
string s = ncratSPrint(f);
print(s);
↦ 2*x*y
```

7.10.6.10 ncratPrint

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncratPrint(f);
f of type ncrat

Return: prints f

Note: print(f);
yields the same result as
ncratPrint(f);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncratPrint(f);
↦ 2*x*y
print(f);
↦ 2*x*y
```

7.10.6.11 ncratFromString

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat f = ncratFromString(s);
s of type string

Return: read string s into ncrat f

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
print(f);
↦ 2*x*y
```

7.10.6.12 ncratFromPoly

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat f = ncratFromPoly(p);
p of type poly

Return: convert poly to ncrat

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
poly p = 2 * x * y;
ncrat f = ncratFromPoly(p);
print(f);
↦ 2*x*y
```

7.10.6.13 ncratPower

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrat h = ncratPower(f, n);
f of type ncrat, n integer

Return: h = f^n

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncrat h = ncratPower(f, 3);
print(h);
↦ 2*x*y*2*x*y*2*x*y
```

7.10.6.14 ncratEvaluateAt

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: matrix M = ncratEvaluateAt(f, vars, point);

Return: Evaluate the ncrat f by substituting in the matrices contained in point for the respective variables contained in var, that is, calculate f(point).

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("x+y");
matrix A[2][2] = 1, 2, 3, 4;
matrix B[2][2] = 5, 6, 7, 8;
matrix M = ncratEvaluateAt(f, list(x, y), list(A, B));
print(M);
↦ 6, 8,
↦ 10,12
```

7.10.6.15 ncrepGet

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep q = ncrepGet(f);
f of type ncrat

Return: q = (u, Q, v) linear representation of f

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncrep q = ncrepGet(f);
print(q);
↦ lvec=
↦ 0,0,0,1
↦
```

```
↦ mat=
↦ 0, 0, 1/2*x,-1/2,
↦ 0, 1, -1/2, 0,
↦ y, -1,0,    0,
↦ -1,0, 0,    0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
```

7.10.6.16 ncrepAdd

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepAdd(q, r);
q, r both of type ncrep

Return: representation s of h = f + g
if q, r are representations of f, g

Note: operator '+' for ncrep is overloaded
with this procedure, hence
ncrep s = q + r;
yields the same result as
ncrep s = ncrepAdd(q, r);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("x");
ncrat g = ncratFromString("y");
ncrep q = ncrepGet(f);
ncrep r = ncrepGet(g);
ncrep s1, s2;
s1 = ncrepAdd(q, r);
print(s1);
↦ lvec=
↦ 0,1,0,1
↦
↦ mat=
↦ x, -1,0, 0,
↦ -1,0, 0, 0,
↦ 0, 0, y, -1,
↦ 0, 0, -1,0
↦
↦ rvec=
↦ 0,
↦ 1,
↦ 0,
↦ 1
s2 = q + r;
print(s2);
↦ lvec=
↦ 0,1,0,1
↦
↦ mat=
↦ x, -1,0, 0,
↦ -1,0, 0, 0,
↦ 0, 0, y, -1,
↦ 0, 0, -1,0
↦
↦ rvec=
↦ 0,
↦ 1,
↦ 0,
↦ 1
```

7.10.6.17 ncrepSubstract

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepSubstract(q, r);
q, r both of type ncrep

Return: representation s of h = f - g
if q, r are representations of f, g

Note: operator '-' for ncrep is overloaded
with this procedure, hence
ncrep s = q - r;
yields the same result as
ncrep s = ncrepSubstract(q, r);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("x");
ncrat g = ncratFromString("y");
ncrep q = ncrepGet(f);
ncrep r = ncrepGet(g);
ncrep s1, s2;
s1 = ncrepSubstract(q, r);
print(s1);
↦ lvec=
↦ 0,1,0,1
↦
↦ mat=
↦ x, -1,0, 0,
↦ -1,0, 0, 0,
↦ 0, 0, -y,1,
↦ 0, 0, 1, 0
↦
↦ rvec=
↦ 0,
↦ 1,
↦ 0,
↦ 1
s2 = q - r;
```

```
print(s2);
↦ lvec=
↦ 0,1,0,1
↦
↦ mat=
↦ x, -1,0, 0,
↦ -1,0, 0, 0,
↦ 0, 0, -y,1,
↦ 0, 0, 1, 0
↦
↦ rvec=
↦ 0,
↦ 1,
↦ 0,
↦ 1
```

7.10.6.18 ncrepMultiply

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepMultiply(q, r);
q, r both of type ncrep

Return: representation s of h = f * g
if q, r are representations of f, g

Note: operator '*' for ncrep is overloaded
with this procedure, hence
ncrep s = q * r;
yields the same result as
ncrep s = ncrepMultiply(q, r);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("x");
ncrat g = ncratFromString("y");
ncrep q = ncrepGet(f);
ncrep r = ncrepGet(g);
ncrep s1, s2;
s1 = ncrepMultiply(q, r);
print(s1);
↦ lvec=
↦ 0,0,0,1
↦
↦ mat=
↦ 0, 0, x, -1,
↦ 0, 1, -1,0,
↦ y, -1,0, 0,
↦ -1,0, 0, 0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
s2 = q * r;
print(s2);
↦ lvec=
↦ 0,0,0,1
↦
↦ mat=
↦ 0, 0, x, -1,
↦ 0, 1, -1,0,
↦ y, -1,0, 0,
↦ -1,0, 0, 0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
```

7.10.6.19 ncrepInvert

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepInvert(q);
q of type ncrep

Return: representation of h = inv(f)
if q is a representation of f

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncrep q = ncrepGet(f);
ncrep s = ncrepInvert(q);
print(s);
↦ lvec=
↦ 1,0,0,0,0
↦
↦ mat=
↦ 0,0, 0, 0,     1,
↦ 0,0, 0, -1/2*x,1/2,
↦ 0,0, -1,1/2,   0,
↦ 0,-y,1, 0,     0,
↦ 1,1, 0, 0,     0
↦
↦ rvec=
↦ 1,
↦ 0,
↦ 0,
↦ 0,
↦ 0
```

7.10.6.20 ncrepPrint

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrepPrint(q);
q of type ncrep

Return: prints q

Note: print(q);
yields the same result as
ncrepPrint(q);

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncrep q = ncrepGet(f);
ncrepPrint(q);
↦ lvec=
↦ 0,0,0,1
↦
↦ mat=
↦ 0, 0, 1/2*x,-1/2,
↦ 0, 1, -1/2, 0,
↦ y, -1,0,    0,
↦ -1,0, 0,    0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
print(q);
↦ lvec=
↦ 0,0,0,1
↦
↦ mat=
↦ 0, 0, 1/2*x,-1/2,
↦ 0, 1, -1/2, 0,
↦ y, -1,0,    0,
↦ -1,0, 0,    0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
```

7.10.6.21 ncrepDim

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrepDim(q);
q of type ncrep

Return: dimension of q;
returns 0 if q represents the zero-function

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("2*x*y");
ncrep q = ncrepGet(f);
print(q);
↦ lvec=
↦ 0,0,0,1
↦
↦ mat=
↦ 0, 0, 1/2*x,-1/2,
↦ 0, 1, -1/2, 0,
↦ y, -1,0,    0,
↦ -1,0, 0,    0
↦
↦ rvec=
↦ 0,
↦ 0,
↦ 0,
↦ 1
ncrepDim(q);
↦ 4
```

7.10.6.22 ncrepSubstitute

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepSubstitute(q, l);
q of type ncrep, vars = (x1, ..., xg),
points = (A1, ... , Ag) with Ai matrices of the
same dimension and xi of type poly are nc variables

Return: substitutes in Ai for xi in q

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("x+y");
ncrep q = ncrepGet(f);
matrix A[2][2] = 1, 2, 3, 4;
matrix B[2][2] = 5, 6, 7, 8;
ncrep s = ncrepSubstitute(q, list(x, y), list(A, B));
print(q);
↦ lvec=
↦ 0,1,0,1
↦
↦ mat=
↦ x, -1,0, 0,
↦ -1,0, 0, 0,
↦ 0, 0, y, -1,
↦ 0, 0, -1,0
↦
↦ rvec=
↦ 0,
↦ 1,
↦ 0,
↦ 1
print(s);
↦ lvec=
↦ 0,0,1,0,0,0,1,0,
↦ 0,0,0,1,0,0,0,1
↦
↦ mat=
↦ 1, 2, -1,0, 0, 0, 0, 0,
↦ 3, 4, 0, -1,0, 0, 0, 0,
↦ -1,0, 0, 0, 0, 0, 0, 0,
↦ 0, -1,0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 5, 6, -1,0,
↦ 0, 0, 0, 0, 7, 8, 0, -1,
↦ 0, 0, 0, 0, -1,0, 0, 0,
↦ 0, 0, 0, 0, 0, -1,0, 0
↦
↦ rvec=
↦ 0,0,
↦ 0,0,
↦ 1,0,
↦ 0,1,
↦ 0,0,
↦ 0,0,
↦ 1,0,
↦ 0,1
```

7.10.6.23 ncrepEvaluate

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: matrix M = ncrepEvaluate(q);

Return: for q=(u, Q, v) calculate -u*Q^(-1)*v

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y", "z"));
ncrat f = ncratFromString("x+y");
ncrep q = ncrepGet(f);
matrix A[2][2] = 1, 2, 3, 4;
matrix B[2][2] = 5, 6, 7, 8;
ncrep s = ncrepSubstitute(q, list(x, y), list(A, B));
matrix M = ncrepEvaluate(s);
print(M);
↦ 6, 8,
↦ 10,12
```

7.10.6.24 ncrepEvaluateAt

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: matrix M = ncrepEvaluateAt(q, vars, point);

Return: For q=(u, Q, v) calculate -u*Q(point)^(-1)*v,
that is to say, evaluate the ncrat represented
by q at point (scalar or matrix point).

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("x+y");
ncrep q = ncrepGet(f);
matrix A[2][2] = 1, 2, 3, 4;
matrix B[2][2] = 5, 6, 7, 8;
matrix M = ncrepEvaluateAt(q, list(x, y), list(A, B));
print(M);
↦ 6, 8,
↦ 10,12
```

7.10.6.25 ncrepIsDefinedDim

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: list l = ncrepIsDefinedDim(q, N, vars, n, maxcoeff);

Return: list(k, list vars, list(A1, ..., Ak)), where:
If k = N then there are matrices A1, ..., Ak of size N such that q is defined at A = (A1, ..., Ak), i.e.,
q.mat is invertible at A.
If k = 0 then no such point was found.

Note: Test whether q.mat is invertible via evaluation
at random matrix points with integer coefficients
in [-maxcoeff, maxcoeff]. Stops after n tries.
Use square matrices of dimension N. The list vars
contains the nc variables which occur in q.

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(x*y-y*x)");
ncrep q = ncrepGet(f);
ncrepIsDefinedDim(q, 1, list(x, y), 10, 100);
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
↦       y
↦ [3]:
↦    empty list
ncrepIsDefinedDim(q, 2, list(x, y), 10, 100);
↦ [1]:
↦    2
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
↦       y
↦ [3]:
↦    [1]:
↦       _[1,1]=-55
↦       _[1,2]=-24
↦       _[2,1]=39
↦       _[2,2]=-17
↦    [2]:
↦       _[1,1]=36
↦       _[1,2]=-58
↦       _[2,1]=-13
↦       _[2,2]=-55
```

7.10.6.26 ncrepIsDefined

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: list l = ncrepIsDefined(q, vars, n, maxcoeff);

Return: list(dim, list vars, list(A1, ..., Ak)), where:
If dim > 0 then there are matrices A1, ..., Ak of size dim such that q is defined at A = (A1, ..., Ak), i.e.,
q.mat is invertible at A.
If dim = 0 then no such point was found.

Note: Test whether q.mat is invertible via evaluation
at random matrix points with integer coefficients
in [-maxcoeff, maxcoeff]. Stops after n tries.
Use ixi-matrix in i-th try. The list vars contains the
nc variables which occur in q.

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(x*y-y*x)");
ncrep q = ncrepGet(f);
ncrepIsDefined(q, list(x, y), 5, 10);
↦ [1]:
↦    2
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
↦       y
↦ [3]:
↦    [1]:
↦       _[1,1]=0
↦       _[1,2]=-9
↦       _[2,1]=-2
↦       _[2,2]=7
↦    [2]:
↦       _[1,1]=8
↦       _[1,2]=-9
↦        _[2,1]=-4
↦        _[2,2]=-2
ncrat g = ncratFromString("inv(x-x)");
ncrep r = ncrepGet(g);
ncrepIsDefined(r, list(x), 5, 10);
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       x
↦ [3]:
↦    empty list
```

7.10.6.27 ncrepIsRegular

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: list l = ncrepIsRegular(q, vars, n, maxcoeff);

Return: list(k, list vars, list(a1, ..., ak)), where:
If k = 1 then there are scalars (1x1-matrices) a1, ..., ak such that q is defined at a = (a1, ..., ak), i.e.,
q.mat is invertible at a.
If k = 0 then no such point was found.

Note: Test whether q.mat is invertible via evaluation
at random integers in [-maxcoeff, maxcoeff].
Stops after n tries. The list vars
contains the nc variables which occur in q.

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(x*y-y*x)");
ncrep q = ncrepGet(f);
ncrepIsRegular(q, list(x, y), 10, 100);
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
↦       y
↦ [3]:
↦    empty list
ncrat g = ncratFromString("inv(1+x*y-y*x)");
ncrep r = ncrepGet(g);
ncrepIsRegular(r, list(x, y), 10, 100);
↦ [1]:
↦    1
↦ [2]:
↦    [1]:
↦       x
↦    [2]:
```

```
↦        y
↦ [3]:
↦    [1]:
↦       _[1,1]=-55
↦    [2]:
↦       _[1,1]=-24
```

7.10.6.28 ncrepRegularZeroMinimize

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepRegularZeroMinimize(q, l);

Return: ncrep s representing the same rational
function as ncrep q, where s is of minimal size

Assumption:
q is regular at zero, i.e.,
if one substitutes in 0 for all nc variables in q then q.mat has to be invertible

Note: list l = list(x1, ..., xn) has to consist
exactly of the nc variables occurring in q

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(1+x*y-y*x)");
ncrep q = ncrepGet(f);
ncrepDim(q);
↦ 11
ncrep s = ncrepRegularZeroMinimize(q, list(x, y));
ncrepDim(s);
↦ 3
s;
↦ lvec=
↦ 0,1,0
↦
↦ mat=
↦ 1, y, 0,
↦ -x,1, -y,
↦ 0, -x,1
↦
↦ rvec=
↦ 0,
↦ -1,
↦ 0
↦
```

7.10.6.29 ncrepRegularMinimize

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep s = ncrepRegularMinimize(q, vars, point);

Return: ncrep s representing the same rational
function as ncrep q, where s is of minimal size

Assumption:
q is regular at scalar point a, i.e.,
if one substitutes in ai for all nc variables xi in q then q.mat has to be invertible

Note: list vars = list(x1, ..., xn) has to consist
exactly of the nc variables occurring in q and
list point = list(a1, ..., an) consists of scalars

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(x*y)");
ncrep q = ncrepGet(f);
ncrepDim(q);
↦ 5
ncrep s = ncrepRegularMinimize(q, list(x, y), list(1, 1));
ncrepDim(s);
↦ 2
s;
↦ lvec=
↦ -1,0
↦
↦ mat=
↦ -y,x+1,
↦ 0, -x
↦
↦ rvec=
↦ 1,
↦ -1
↦
```

7.10.6.30 ncrepGetRegularZeroMinimal

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep q = ncrepGetRegularZeroMinimal(f, vars);

Return: q is a representation of f with
minimal dimension

Assumption:
f is regular at zero, i.e.,
f(0) has to be defined

Note: list vars = list(x1, ..., xn) has to consist
exactly of the nc variables occurring in f

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(1+x*y-y*x)");
list vars = list(x, y);
ncrep q = ncrepGetRegularZeroMinimal(f, vars);
q;
↦ lvec=
↦ 0,1,0
```

```
↦
↦ mat=
↦ 1, y, 0,
↦ -x,1, -y,
↦ 0, -x,1
↦
↦ rvec=
↦ 0,
↦ -1,
↦ 0
↦
```

7.10.6.31 ncrepGetRegularMinimal

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: ncrep q = ncrepGetRegularMinimal(f, vars, point);

Return: q is a representation of f with
minimal dimension

Assumption:
f is regular at point, i.e.,
f(point) has to be defined

Note: list vars = list(x1, ..., xn) has to consist
exactly of the nc variables occurring in f and
list point = (p1, ..., pn) of scalars such that
f(point) is defined

Example:

```
LIB "ncrat.lib";
// We want to prove the Hua's identity, telling that for two
// invertible elements x,y from a division ring, one has
// inv(x+x*inv(y)*x)+inv(x+y) = inv(x)
// where inv(t) stands for the two-sided inverse of t
ncInit(list("x", "y"));
ncrat f = ncratFromString("inv(x+x*inv(y)*x)+inv(x+y)-inv(x)");
print(f);
↦ inv(x+x*inv(y)*x)+inv(x+y)-inv(x)
ncrep r = ncrepGet(f);
ncrepDim(r);
↦ 18
ncrep s = ncrepGetRegularMinimal(f, list(x, y), list(1, 1));
ncrepDim(s);
↦ 0
print(s);
↦ lvec=
↦ 0
↦
↦ mat=
↦ 1
↦
↦ rvec=
↦ 0
// since s represents the zero element, Hua's identity holds.
```

7.10.6.32 ncrepPencilGet

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: list pencil = ncrepPencilGet(r, vars);

Return: pencil = list(vars, matrices),
where vars = list(1, x1, ..., xg) are the variables
occurring in r and matrices = (Q0, ..., Qg) is a list of matrices such that
r.mat = Q0 * x0 + ... + Qg * xg
with x0 = 1

Note: list vars = list(x1, ..., xn) has to consist
exactly of the nc variables occurring in f

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("x*y");
ncrep r = ncrepGet(f);
print(r.mat);
↦ 0, 0, x, -1,
↦ 0, 1, -1,0,
↦ y, -1,0, 0,
↦ -1,0, 0, 0
list l = ncrepPencilGet(r, list(x, y));
print(l[1]);
↦ [1]:
↦    1
↦ [2]:
↦    x
↦ [3]:
↦    y
print(l[2][1]);
↦ 0, 0, 0, -1,
↦ 0, 1, -1,0,
↦ 0, -1,0, 0,
↦ -1,0, 0, 0
print(l[2][2]);
↦ 0,0,1,0,
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
print(l[2][3]);
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 1,0,0,0,
↦ 0,0,0,0
```

7.10.6.33 ncrepPencilCombine

Procedure from library `ncrat.lib` (see Section 7.10.6 [ncrat_lib], page 675).

Usage: matrix Q = ncrepPencilCombine(pencil);

Return: matrix Q = Q0*x0 + ... + Qg*xg,
where vars = list(x0, ..., xg) consists of polynomials and matrices = (Q0, ..., Qg) is a list of matrices

Example:

```
LIB "ncrat.lib";
ncInit(list("x", "y"));
ncrat f = ncratFromString("x*y");
ncrep r = ncrepGet(f);
print(r.mat);
↦ 0, 0, x, -1,
↦ 0, 1, -1,0,
↦ y, -1,0, 0,
↦ -1,0, 0, 0
list l = ncrepPencilGet(r, list(x, y));
matrix Q = ncrepPencilCombine(l);
print(Q);
↦ 0, 0, x, -1,
↦ 0, 1, -1,0,
↦ y, -1,0, 0,
↦ -1,0, 0, 0
```

7.11 Release Notes (letterplace)

NEWS in SINGULAR:LETTERPLACE 4.3.2

News for SINGULAR:LETTERPLACE version 4.3.2

New functions:

added support for free bimodules of a fixed rank (Section 7.8.3 [freeAlgebra (letterplace)], page 626, Section 7.8.8 [ncgen], page 630)

several types of monomial orderings become available, among them three types of elimination orderings

twostd (Section 7.8.14 [twostd (letterplace)], page 634), reduce (Section 7.8.9 [reduce (letterplace)], page 631) and other functions support subbimodules

syz (Section 7.8.13 [syz (letterplace)], page 633), lift (Section 7.8.5 [lift (letterplace)], page 628), liftstd (Section 7.8.6 [liftstd (letterplace)], page 629), modulo (Section 7.8.7 [modulo (letterplace)], page 630) implemented

bracket (Section 7.3.2 [bracket], page 335) and maxideal (Section 5.1.88 [maxideal], page 219) work in Letterplace

the options `redSB`, `redTail` are effective for computations related to Groebner bases

the options `prot`, `mem` are effective for the whole Letterplace subsystem

New libraries:

fpaprops_lib: Algorithms for properties of quotient algebras (Section 7.10.3 [fpaprops_lib], page 659) ncHilb.lib: Hilbert functions for non-commutative algebras (Section 7.10.5 [ncHilb_lib], page 672)

Changed libraries:

fpadim.lib: Vector space dimension, basis and Hilbert series for finitely presented algebras (Section 7.10.1 [fpadim_lib], page 639), numerous enhancements, partially implemented in the kernel

freegb.lib: Main initialization and convenience tools (Section 7.10.4 [freegb_lib], page 665)

Changes in the kernel/build system:

SINGULAR:LETTERPLACE is available as the dynamical module

adaptions/functions for `Singular.jl`(`https://github.com/oscar-system/Singular.jl`)

News for SINGULAR:LETTERPLACE version 4-1-2

New libraries:

fpalgebras.lib: Generation of various algebras in the letterplace case (Section 7.10.2 [fpalgebras_lib], page 645)

ncrat.lib: Manipulating non-commutative rational functions (Section 7.10.6 [ncrat_lib], page 675)

Changed/updated libraries:

freegb.lib: lpDivision, lpPrint (Section 7.10.4 [freegb_lib], page 665)

fpadim.lib (Section 7.10.1 [fpadim_lib], page 639)

ncfactor.lib (Section 7.5.12 [ncfactor_lib], page 486) is available for Letterplace rings

Changes in the kernel/build system:

code for free algebras (letterplace rings) rewritten (using now the standard `+,-,*,^,std`,...) (Section 7.7 [LETTERPLACE], page 616)

new command `rightstd` (Section 7.8.10 [rightstd (letterplace)], page 632)

extended `twostd` to Letterplace (Section 7.8.14 [twostd (letterplace)], page 634, Section 7.3.29 [twostd (plural)], page 363)

Appendix A Examples

A.1 Programming

A.1.1 Basic programming

We show in the example below the following:

- define the ring R of characteristic 32003, variables x,y,z, monomial ordering dp (implementing F_32003[x,y,z])
- list the information about R by typing its name
- check the order of the variables
- define the integers a,b,c,t
- define a polynomial f (depending on a,b,c,t) and display it
- define the jacobian ideal i of f
- compute a Groebner basis of i
- compute the dimension of the algebraic set defined by i (requires the computation of a Groebner basis)
- create and display a string in order to comment the result (text between quotes " "; is a 'string')
- load a library (see Section D.4.26 [primdec_lib], page 1239)
- compute a primary decomposition for i and assign the result to a list L (which is a list of lists of ideals)
- display the number of primary components and the first primary and prime components (entries of the list L[1])
- implement the localization of F_32003[x,y,z] at the homogeneous maximal ideal (generated by x,y,z) by defining a ring with local monomial ordering (ds in place of dp)
- map i to this ring (see Section 5.1.59 [imap], page 198) - we may use the same name i, since ideals are ring dependent data
- compute the local dimension of the algebraic set defined by i at the origin (= dimension of the ideal generated by i in the localization)
- compute the local dimension of the algebraic set defined by i at the point (-2000,-6961,-7944) (by applying a linear coordinate transformation)

For a more basic introduction to programming in Singular, we refer to Section 2.3 [Getting started], page 6.

```
ring R = 32003,(x,y,z),dp;
R;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
x > y;
↦ 1
y > z;
↦ 1
```

```
int a,b,c,t = 1,2,-1,4;
poly f = a*x3+b*xy3-c*xz3+t*xy2z2;
f;
↦ 4xy2z2+2xy3+xz3+x3
ideal i = jacob(f);    // Jacobian Ideal of f
ideal si = std(i);     // compute Groebner basis
int dimi = dim(si);
string s = "The dimension of V(i) is "+string(dimi)+".";
s;
↦ The dimension of V(i) is 1.
LIB "primdec.lib";     // load library primdec.lib
list L = primdecGTZ(i);
size(L);               // number of prime components
↦ 6
L[1][1];               // first primary component
↦ _[1]=2y2z2+y3-16001z3
↦ _[2]=x
L[1][2];               // corresponding prime component
↦ _[1]=2y2z2+y3-16001z3
↦ _[2]=x
ring Rloc = 32003,(x,y,z),ds;   // ds = local monomial ordering
ideal i = imap(R,i);
dim(std(i));
↦ 1
map phi = R, x-2000, y-6961, z-7944;
dim(std(phi(i)));
↦ 0
```

A.1.2 Writing procedures and libraries

The user may add their own commands to the commands already available in Singular by writing Singular procedures. There are basically two kinds of procedures:

- procedures written in the Singular programming language (which are usually collected in Singular libraries).
- procedures written in C/C++ (collected in dynamic modules).

At this point, we restrict ourselves to describing the first kind of (library) procedures, which are sufficient for most applications. The syntax and general structure of a library (procedure) is described in Section 3.7 [Procedures], page 50, and Section 3.8 [Libraries], page 55.

The probably most efficient way of writing a new library is to use one of the official Singular libraries, say `ring.lib` as a sample. On a Unix-like operating system, type `LIB "ring.lib";` to get information on where the libraries are stored on your disk.

Singular provides several commands and tools, which may be useful when writing a procedure, for instance, to have a look at intermediate results (see Section 3.9 [Debugging tools], page 68).

If such a library should be contributed to Singular some formal requirements are needed:

the library header must explain the purpose of the library and (for non-trivial algorithm) a pointer to the algorithm (text book, article, etc.)

all global procedures must have a help string and an example which shows its usage.

it is strongly recommend also to provide test scripts which test the functionality: one should test the essential functionality of the library/command in a relatively short time (say, in no more than 30s), other tests should check the functionality of the library/command in detail so that, if possible, all relevant cases/results are tested. Nevertheless, such a test should not run longer than, say, 10 minutes.

We give short examples of procedures to demonstrate the following:

- Write procedures which return an integer (ring independent), see also Section A.4.1 [Milnor and Tjurina number], page 734. (Here we restrict ourselves to the main body of the procedures).
  - The procedure `milnorNumber` must be called with one parameter, a polynomial. The name g is local to the procedure and is killed automatically when leaving the procedure. `milnorNumber` returns the Milnor number (and displays a comment).
  - The procedure `tjurinaNumber` has no specified number of parameters. Here, the parameters are referred to by `#[1]` for the 1st, `#[2]` for the 2nd parameter, etc. `tjurinaNumber` returns the Tjurina number (and displays a comment).
  - the procedure `milnor_tjurina` which returns a list consisting of two integers, the Milnor and the Tjurina number.
- Write a procedure which creates a new ring and returns data dependent on this new ring (two numbers) and an int. In this example, we also show how to write a help text for the procedure (which is optional, but recommended).

```
proc milnorNumber (poly g)
{
   "Milnor number:";
   return(vdim(std(jacob(g))));
}

proc tjurinaNumber
{
   "Tjurina number:";
   return(vdim(std(jacob(#[1])+#[1])));
}

proc milnor_tjurina (poly f)
{
   ideal j=jacob(f);
   list L=vdim(std(j)),vdim(std(j+f));
   return(L);
}

proc real_sols (number b, number c)
"USAGE: real_sols (b,c);  b,c number
ASSUME: active basering has characteristic 0
RETURN: list: first entry is an integer (the number of different real
        solutions). If this number is non-negative, the list has as second
        entry a ring in which the list SOL of real solutions of x^2+bx+c=0
        is stored (as floating point number, precision 30 digits).
NOTE:   This procedure calls laguerre_solve from solve.lib.
"
{
  def oldring = basering;  // assign name to the ring active when
                           // calling the procedure
  number disc = b^2-4*c;
  if (disc>0) { int n_of_sols = 2; }
  if (disc==0) { int n_of_sols = 1; }
  string s = nameof(var(1));  // name of first ring variable
  if (disc>=0) {
    execute("ring rinC =(complex,30),("+s+"),lp;");
    if (not(defined(laguerre_solve))) { LIB "solve.lib"; }
    poly f = x2+imap(oldring,b)*x+imap(oldring,c);
                        // f is a local ring-dependent variable
    list SOL = laguerre_solve(f,30);
    export SOL;         // make SOL a global ring-dependent variable
                        // such variables are still accessible when the
                        // ring is among the return values of the proc
    setring oldring;
    return(list(n_of_sols,rinC));
  }
  else {
    return(list(0));
  }
}

//
// We now apply the procedures which are defined by the
// lines of code above:
//
ring r = 0,(x,y),ds;
poly f = x7+y7+(x-y)^2*x2y2;

milnorNumber(f);
↦ Milnor number:
↦ 28
tjurinaNumber(f);
↦ Tjurina number:
↦ 24
milnor_tjurina(f);      // a list containing Milnor and Tjurina number
↦ [1]:
↦    28
↦ [2]:
↦    24

def L=real_sols(2,1);
L[1];                   // number of real solutions of x^2+2x+1
↦ 1
def R1=L[2];
setring R1;
listvar(R1);            // only global ring-dependent objects are still alive
↦ // R1                          [0]  *ring
↦ // SOL                         [0]  list, size: 2
SOL;                    // the real solutions
↦ [1]:
↦ -1
↦ [2]:
↦ -1

setring r;
L=real_sols(1,1);
L[1];                   // number of reals solutions of x^2+x+1
```

```
↦ 0

setring r;
L=real_sols(1,-5);
L[1];                   // number of reals solutions of x^2+x-5
↦ 2
def R3=L[2];
setring R3; SOL;        // the real solutions
↦ [1]:
↦ -2.79128784747792000329402359686
↦ [2]:
↦ 1.79128784747792000329402359686
```

Writing a dynamic module is not as simple as writing a library procedure, since it does not only require some knowledge of C/C++, but also about the way the Singular kernel works. See also Section A.1.9 [Dynamic modules], page 708.

A.1.3 Rings associated to monomial orderings

In Singular we may implement localizations of the polynomial ring by choosing an appropriate monomial ordering (when defining the ring by the `ring` command). We refer to Section B.2 [Monomial orderings], page 768 for a thorough discussion of the monomial orderings available in Singular.

At this point, we restrict ourselves to describing the relation between a monomial ordering and the ring (as mathematical object) which is implemented by the ordering. This is most easily done by describing the set of units: if `>` is a monomial ordering then precisely those elements which have leading monomial 1 are considered as units (in all computations performed with respect to this ordering).

In mathematical terms: choosing a monomial ordering `>` implements the localization of the polynomial ring with respect to the multiplicatively closed set of polynomials with leading monomial 1.

That is, choosing $>$ implements the ring

$$K[x]_> := \left\{ \frac{f}{u} \;\middle|\; f, u \in K[x],\, LM(u) = 1 \right\}.$$

If `>` is global (that is, 1 is the smallest monomial), the implemented ring is just the polynomial ring. If `>` is local (that is, if 1 is the largest monomial), the implemented ring is the localization of the polynomial ring w.r.t. the homogeneous maximal ideal. For a mixed ordering, we obtain "something in between these two rings":

```
ring R = 0,(x,y,z),dp;      // polynomial ring (global ordering)
poly f = y4z3+2x2y2z2+4z4+5y2+1;
f;                          // display f in a degrevlex-ordered way
↦ y4z3+2x2y2z2+4z4+5y2+1
short=0;                    // avoid short notation
f;
↦ y^4*z^3+2*x^2*y^2*z^2+4*z^4+5*y^2+1
short=1;
leadmonom(f);               // leading monomial
↦ y4z3

ring r = 0,(x,y,z),ds;      // local ring (local ordering)
```

```
poly f = fetch(R,f);
f;                           // terms of f sorted by degree
↦ 1+5y2+4z4+2x2y2z2+y4z3
leadmonom(f);                // leading monomial
↦ 1

// Now we implement more "advanced" examples of rings:
//
// 1)   (K[y]_<y>)[x]
//
int n,m=2,3;
ring A1 = 0,(x(1..n),y(1..m)),(dp(n),ds(m));
poly f  = x(1)*x(2)^2+1+y(1)^10+x(1)*y(2)^5+y(3);
leadmonom(f);
↦ x(1)*x(2)^2
leadmonom(1+y(1));           // unit
↦ 1
leadmonom(1+x(1));           // no unit
↦ x(1)

//
// 2)  some ring in between (K[x]_<x>)[y] and K[x,y]_<x>
//
ring A2 = 0,(x(1..n),y(1..m)),(ds(n),dp(m));
leadmonom(1+x(1));         // unit
↦ 1
leadmonom(1+x(1)*y(1));    // unit
↦ 1
leadmonom(1+y(1));         // no unit
↦ y(1)

//
// 3)  K[x,y]_<x>
//
ring A4 = (0,y(1..m)),(x(1..n)),ds;
leadmonom(1+y(1));         // in ground field
↦ 1
leadmonom(1+x(1)*y(1));    // unit
↦ 1
leadmonom(1+x(1));         // unit
↦ 1
```

Note, that even if we implicitly compute over the localization of the polynomial ring, most computations are explicitly performed with polynomial data only. In particular, `1/(1-x);` does not return a power series expansion or a fraction but 0 (division with remainder in polynomial ring).

See Section 5.1.26 [division], page 174 for division with remainder in the localization and Section D.6.17.2 [invunit], page 1738 for a procedure returning a truncated power series expansion of the inverse of a unit.

A.1.4 Long coefficients

The following innocent example produces in its standard basis extremely long coefficients in char 0 for the lexicographical ordering. But a very small deformation does not (the undeformed example is degenerated with respect to the Newton boundary). This example demonstrates that it might be wise, for complicated examples, to do the calculation first in positive char (e.g., 32003). It has been shown, that in complicated examples, more than 95 percent of the time needed for a standard basis computation is used in the computation of the coefficients (in char 0). The representation of long integers with real is demonstrated.

```
timer = 1;                             // activate the timer
ring R0 = 0,(x,y),lp;
poly f = x5+y11+xy9+x3y9;
ideal i = jacob(f);
ideal i1 = i,i[1]*i[2];                // undeformed ideal
ideal i2 = i,i[1]*i[2]+1/1000000*x5y8;  // deformation of i1
i1; i2;
↦ i1[1]=5x4+3x2y9+y9
↦ i1[2]=9x3y8+9xy8+11y10
↦ i1[3]=45x7y8+27x5y17+45x5y8+55x4y10+36x3y17+33x2y19+9xy17+11y19
↦ i2[1]=5x4+3x2y9+y9
↦ i2[2]=9x3y8+9xy8+11y10
↦ i2[3]=45x7y8+27x5y17+45000001/1000000x5y8+55x4y10+36x3y17+33x2y19+9xy17+1\
   1y19
ideal j = std(i1);
j;
↦ j[1]=264627y39+26244y35-1323135y30-131220y26+1715175y21+164025y17+1830125\
   y16
↦ j[2]=121039477919718467198383218863933929137500650608 75xy8-28639152114168\
   319870133193925000326676773863287 5y38-31954402206909026926764622877573565\
   78554430672591y37+574366214208226638497213812657388952828 46320y36+1657764\
   2149487994975739182100310673539324394 00y35+21301848158930819119567722389 8\
   98682697001205500y34+182219415866306656558599197696156571964806980 6148y33\
   -4701709279892816135156972313196394005220175y32-13518722696881922676007 86\
   97600850686824231975y31-387306330592981081696151697602503805300114137 5y30\
   +13258866758438740479903820054211440618612900800 00y29+1597720195476063141\
   94679458955424060895269668873 10y28-2627018133630909266063334800262533042 6\
   7126525y27-758608269089333526902713624894485954472795312 5y26-867853074106\
   494646022858433516721489653959456 25y25-554580814327359410217325233115183 5\
   700278863924745y24+1907556301346043736467915377903839489563832 5y23+548562\
   32271550176105834899677692256107402112 5y22+157465452677648386073957464715\
   68100780933983125y21-14142791297211762229786542358173595055551911562 50y20\
   -2071119006944589361521339965003571537816994342312 5y19+272942733337472665\
   5734180929779053229840097 50y18+789065115845334505801847294677413365720955\
   3750y17+635548970384916867877296560610447246510898031 25y16-22099251729923\
   906699732244761028266074350255961625y14+147937139679655904353579489722585\
   91339027857296625y10
↦ j[3]=5x4+3x2y9+y9
// Compute average coefficient length (=51) by
//   - converting j[2] to a string in order to compute the number
//   of characters
//   - divide this by the number of monomials:
size(string(j[2])) div size(j[2]);
↦ 51
```

```
vdim(j);
↦ 63
// For a better representation normalize the long coefficients
// of the polynomial j[2] and map it to real:
poly p=(1/12103947791971846719838321886393392913750065060875)*j[2];
ring R1=real,(x,y),lp;
short=0; // force the long output format
poly p=imap(R0,p);
p;
↦ x*y^8-(2.366e-02)*y^38-(2.640e-01)*y^37+(4.745e-06)*y^36+(1.370e-04)*y^35\
  +(1.760e-03)*y^34+(1.505e-01)*y^33-(3.884e-07)*y^32-(1.117e-05)*y^31-(3.2\
  00e-04)*y^30+(1.095e-01)*y^29+(1.320e+00)*y^28-(2.170e-05)*y^27-(6.267e-0\
  4)*y^26-(7.170e-03)*y^25-(4.582e-01)*y^24+(1.576e-06)*y^23+(4.532e-05)*y^\
  22+(1.301e-03)*y^21-(1.168e-01)*y^20-(1.711e+00)*y^19+(2.255e-05)*y^18+(6\
  .519e-04)*y^17+(5.251e-03)*y^16-(1.826e+00)*y^14+(1.222e+00)*y^10
// Compute a standard basis for the deformed ideal:
setring R0; // return to the original ring R0
j = std(i2);
j;
↦ j[1]=y16
↦ j[2]=65610xy8+17393508y27+7223337y23+545292y19+6442040y18-119790y14+80190\
  y10
↦ j[3]=5x4+3x2y9+y9
vdim(j);
↦ 40
```

A.1.5 Parameters

Let us deform the ideal in Section A.1.4 [Long coefficients], page 704 by introducing a parameter t and compute over the ground field Q(t). We compute the dimension at the generic point, i.e., $dim_{Q(t)} Q(t)[x,y]/j$. (This gives the same result as for the deformed ideal above. Hence, the above small deformation was "generic".)

For almost all $a \in Q$ this is the same as $dim_Q Q[x,y]/j_0$, where $j_0 = j|_{t=a}$.

```
  ring Rt = (0,t),(x,y),lp;
  Rt;
↦ // coefficients: QQ(t)
↦ // number of vars : 2
↦ //        block   1 : ordering lp
↦ //                  : names    x y
↦ //        block   2 : ordering C
  poly f = x5+y11+xy9+x3y9;
  ideal i = jacob(f);
  ideal j = i,i[1]*i[2]+t*x5y8;  // deformed ideal, parameter t
  vdim(std(j));
↦ 40
  ring R=0,(x,y),lp;
  ideal i=imap(Rt,i);
  int a=random(1,30000);
  ideal j=i,i[1]*i[2]+a*x5y8;  // deformed ideal, fixed integer a
  vdim(std(j));
↦ 40
```

A.1.6 Formatting output

We show how to insert the result of a computation inside a text by using strings. First we compute the powers of 2 and comment the result with some text. Then we do the same and give the output a nice format by computing and adding appropriate space.

```
  // The powers of 2:
  int  n;
  for (n = 2; n <= 128; n = n * 2)
  {"n = " + string (n);}
↦ n = 2
↦ n = 4
↦ n = 8
↦ n = 16
↦ n = 32
↦ n = 64
↦ n = 128
  // The powers of 2 in a nice format
  int j;
  string space = "";
  for (n = 2; n <= 128; n = n * 2)
  {
    space = "";
    for (j = 1; j <= 5 - size (string (n)); j = j+1)
    { space = space + " "; }
    "n =" + space + string (n);
  }
↦ n =    2
↦ n =    4
↦ n =    8
↦ n =   16
↦ n =   32
↦ n =   64
↦ n =  128
```

A.1.7 Cyclic roots

We write a procedure returning a string that enables us to create automatically the ideal of cyclic roots over the basering with n variables. The procedure assumes that the variables consist of a single letter each (hence no indexed variables are allowed; the procedure `cyclic` in `polylib.lib` does not have this restriction). Then we compute a standard basis of this ideal and some numerical information. (This ideal is used as a classical benchmark for standard basis computations).

```
// We call the procedure 'cyclic':
proc cyclic (int n)
{
   string vs = varstr(basering)+varstr(basering);
   int c=find(vs,",");
   while ( c!=0 )
   {
      vs=vs[1,c-1]+vs[c+1,size(vs)];
      c=find(vs,",");
   }
   string t,s;
   int i,j;
```

```
      for ( j=1; j<=n-1; j=j+1 )
      {
         t="";
         for ( i=1; i <=n; i=i+1 )
         {
            t = t + vs[i,j] + "+";
         }
         t = t[1,size(t)-1] + ","+newline;
         s=s+t;
      }
      s=s+vs[1,n]+"-1";
      return (s);
   }

   ring r=0,(a,b,c,d,e),lp;         // basering, char 0, lex ordering
   string sc=cyclic(nvars(basering));
   sc;                              // the string of the ideal
   ↦ a+b+c+d+e,
   ↦ ab+bc+cd+de+ea,
   ↦ abc+bcd+cde+dea+eab,
   ↦ abcd+bcde+cdea+deab+eabc,
   ↦ abcde-1
   execute("ideal i="+sc+";");      // this defines the ideal of cyclic roots
   i;
   ↦ i[1]=a+b+c+d+e
   ↦ i[2]=ab+bc+cd+ae+de
   ↦ i[3]=abc+bcd+abe+ade+cde
   ↦ i[4]=abcd+abce+abde+acde+bcde
   ↦ i[5]=abcde-1
   timer=1;
   ideal j=std(i);
   ↦ //used time: 7.5 sec
   size(j);                         // number of elements in the std basis
   ↦ 11
   degree(j);
   ↦ // codimension = 5
   ↦ // dimension   = 0
   ↦ // degree      = 70
```

A.1.8 Parallelization with ssi links

In this example, we demonstrate how ssi links can be used to parallelize computations.

To compute a standard basis for a zero-dimensional ideal in the lexicographical ordering, one of the two powerful routines **stdhilb** (see Section 5.1.153 [stdhilb], page 273) and **stdfglm** (see Section 5.1.152 [stdfglm], page 272) should be used. However, in general one cannot predict which one of the two commands is faster. This very much depends on the (input) example. Therefore, we use ssi links to let both commands work on the problem independently and in parallel, so that the one which finishes first delivers the result.

The example we use is the so-called "omndi example". See *Tim Wichmann; Der FGLM-Algorithmus: verallgemeinert und implementiert in Singular; Diplomarbeit Fachbereich Mathematik, Universitaet Kaiserslautern; 1997* for more details.

```
   ring r=0,(a,b,c,u,v,w,x,y,z),lp;
```

```
ideal i=a+c+v+2x-1, ab+cu+2vw+2xy+2xz-2/3,  ab2+cu2+2vw2+2xy2+2xz2-2/5,
ab3+cu3+2vw3+2xy3+2xz3-2/7, ab4+cu4+2vw4+2xy4+2xz4-2/9, vw2+2xyz-1/9,
vw4+2xy2z2-1/25, vw3+xyz2+xy2z-1/15, vw4+xyz3+xy3z-1/21;

link l_hilb,l_fglm = "ssi:fork","ssi:fork";         // 1.

open(l_fglm); open(l_hilb);

write(l_hilb, quote(stdhilb(i)));                        // 2.
write(l_fglm, quote(stdfglm(eval(i))));

list L=list(l_hilb,l_fglm);                              // 3.
int l_index=waitfirst(L);

if (l_index==1)
{
  "stdhilb won !!!!"; size(read(L[1]));
  close(L[1]); close(L[2]);
}
else                                                     // 4.
{
  "stdfglm won !!!!"; size(read(L[2]));
  close(L[1]); close(L[2]);
}
↦ stdfglm won !!!!
↦ 9
```

Some explanatory remarks are in order:

1. Instead of using links of the type `ssi:fork`, we alternatively could use `ssi:tcp` links such that the two "competing" Singular processes run on different machines. This has the advantage of "true" parallel computing since no resource sharing is involved (as it usually is with forked processes).
2. Notice how quoting is used in order to prevent local evaluation (i.e., local computation of results). Since we "forked" the two competing processes, the identifier `i` is defined and has identical values in both child processes. Therefore, the innermost `eval` can be omitted (as is done for the `l_hilb` link), and only the identifier `i` needs to be communicated to the children. However, when `ssi:tcp` links are used, the inner evaluation must be applied so that actual values, and not the identifiers are communicated (as is done for the `l_fglm` link in our example).
3. We wait until one of the two children finished the computation. The main process sleeps (i.e., suspends its execution) in the intermediate time.
4. The child which has won delivers the result and is terminated with the usual `close` command. The other child which is still computing needs to be terminated by an explicit (i.e., system) kill command if running on a different computer. For ssi:fork a `close` is sufficient.

A.1.9 Dynamic modules

The purpose of the following example is to illustrate the use of dynamic modules. Giving an example on how to write a dynamic module is beyond the scope of this manual. A technical reference is given at `https://www.singular.uni-kl.de/Manual/modules.pdf`.

In this example, we use a dynamic module, residing in the file `kstd.so`, which allows ignoring all but the first j entries of vectors when forming the pairs in the standard basis computation.

```
ring r=0,(x,y),dp;
module mo=[x^2-y^2,1,0,0],[xy+y^2,0,1,0],[y^2,0,0,1];
print(mo);
↦ x2-y2,xy+y2,y2,
↦ 1,    0,    0,
↦ 0,    1,    0,
↦ 0,    0,    1

// load dynamic module - at the same time creating package Kstd
// procedures will be available in the packages Top and Kstd
LIB("partialgb.so");
listvar(package);
↦ // Partialgb                      [0]  package Partialgb (C,partialgb.so)
↦ // Singmathic                     [0]  package Singmathic (C,singmathic.s\
   o)
↦ // Standard                       [0]  package Standard (S,standard.lib)
↦ // Top                            [0]  package Top (T)

// set the number of components to be considered to 1
module mostd=partialStd(mo,1);        // calling procedure in Top
                    // obviously computation ignored pairs with leading
                    // term in the second entry
print(mostd);
↦ 0,  0,  y2,xy,x2,
↦ -y, y,  0, 0, 1,
↦ x-y,-x, 0, 1, 0,
↦ 0,  x+y,1, -1,1

// now consider 2 components
module mostd2=Partialgb::partialStd(mo,2); // calling procedure in Partialgb
                    // this time the previously unconsidered pair was
                    // treated too
print(mostd2);
↦ 0,  0,   y2,xy,x2,
↦ 0,  y,   0, 0, 1,
↦ -y, -x+y,0, 1, 0,
↦ x+y,0,   1, -1,1
```

A.2 Computing Groebner and Standard Bases

Several operations with ideals resp. modules uses Groebner bases to compute their result. Most allow an optional string argument to select the algorithm. The possible arguments for the algorithm are

`default`

`std` see Section 5.1.151 [std], page 271

`slimgb` see Section 5.1.145 [slimgb], page 265

`sba` see Section 5.1.140 [sba], page 259; not for module operations

`singmatic` Requires `singmatic.so`

`groebner` see Section 5.1.53 [groebner], page 191

`modstd` see Section D.4.16.1 [modStd], page 1146. Requires Section D.4.16 [modstd_lib], page 1146 `ffmod` see Section D.4.9.7 [ffmodStd], page 1086, Requires Section D.4.9 [ffmodstd_lib], page 1080

`nfmod` see Section D.4.20.2 [nfmodStd], page 1176. Requires Section D.4.20 [nfmodstd_lib], page 1174

`std:sat` Uses `satstd` instead of `std`, see Section D.14.3.2 [satstd], page 2238. Requires Section D.14.3 [customstd_lib], page 2237

Functions with such a choice of the algorithm:

Section 5.1.28 [eliminate], page 176

Section 5.1.65 [intersect], page 202

Section 5.1.94 [modulo], page 223

Section 5.1.81 [liftstd], page 212

Section 5.1.156 [syz], page 280

A.2.1 groebner and std

The basic version of Buchberger's algorithm leaves a lot of freedom in carrying out the computational process. Considerable improvements are obtained by implementing criteria for reducing the number of S-polynomials to be actually considered (e.g., by applying the product criterion or the chain criterion). We refer to Cox, Little, and O'Shea [1997], Chapter 2 for more details and references on these criteria and on further strategies for improving the performance of Buchberger's algorithm (see also Greuel, Pfister [2002]).

Singular's implementation of Buchberger's algorithm is available via the `std` command ('std' referring to `standard` basis). The computation of reduced Groebner and standard bases may be forced by setting `option(redSB)` (see Section 5.1.111 [option], page 234).

However, depending on the monomial ordering of the active basering, it may be advisable to use the `groebner` command instead. This command is provided by the Singular library `standard.lib` which is automatically loaded when starting a Singular session. Depending on some heuristics, `groebner` either refers to the `std` command (e.g., for rings with ordering `dp`), or to one of the algorithms described in the sections Section A.2.2 [Groebner basis conversion], page 712, Section A.2.3 [slim Groebner bases], page 714. For information on the heuristics behind `groebner`, see the library file `standard.lib` (see also Section 2.3.3 [Procedures and libraries], page 10).

We apply the commands `std` and `groebner` to compute a lexicographic Groebner basis for the ideal of cyclic roots over the basering with 6 variables (see Section A.1.7 [Cyclic roots], page 706). We set `option(prot)` to make Singular display some information on the performed computations (see Section 5.1.111 [option], page 234 for an interpretation of the displayed symbols). For long running computations, it is always recommended to set this option.

```
LIB "polylib.lib";
ring r=32003,(a,b,c,d,e,f),lp;
ideal I=cyclic(6);
option(prot);
int t=timer;
system("--ticks-per-sec", 100);         // give time in 1/100 sec
ideal sI=std(I);
↦ [1048575:2]1(5)s2(4)s3(3)s4s(4)s5(6)s(9)s(11)s(14)s(17)-s6s(19)s(21)s(24)\
  s(27)s(30)s(33)s(35)s(38)s(41)ss(42)-s----s7(41)s(43)s(46)s(48)s(51)s(54)\
  s(56)s(59)s(62)s(63)s(65)s(66)s(68)s(70)s(73)s(75)s(78)---ss(81)---------\
  --s(73)--------8-s(66)s(69)s(72)s(75)s(77)s(80)s(81)s(83)s(85)s(88)s(91)s\
  (93)s(96)s(99)s(102)s(105)s(107)s(110)s(113)-------------(100)-----------\
  s(101)s(108)s(110)----------(100)--------9-s(94)s(97)s(99)s(84)s(74)s(77)\
  s(80)---ss(83)s(86)s(73)s(76)s10(78)s(81)s(82)s(84)s(86)s(89)s(92)s(94)s(\
  97)s(100)s(103)s(82)s(84)s(86)s(89)s(92)s(95)s11(98)s(87)s(90)s(93)s(95)s\
  (98)s(101)s(104)----(100)---12-s(99)s(90)s(93)s(92)---------s(86)--------\
  ---13-s(74)s(77)s(79)s(82)s(85)s(88)------------------14-s(64)s(67)ss(70)\
  s(73)s(77)s(81)-----------------------15-s(57)s(65)s(68)ss(71)-----------\
  ------s(57)----16-s(55)ss(56)------------------------17-s(34)s(32)-------\
  -18-s(26)s(28)s-----19-s(25)s(28)s(31)------20-s(27)s(30)s(35)-------21-s\
  (23)s(26)------22-s(22)------23-s(15)24-s(17)-s(19)--25-s(18)s(19)s26-s(2\
  1)---------27-s(11)28-s(13)--29-s(12)-30--s--31-s(11)---32-s33(7)s(10)---\
  34-s-35----36-s37(6)s38s39s40---42-s43(5)s44s45--48-s49s50s51---54-s55(4)\
  --67-86-
↦ product criterion:664 chain criterion:2844
timer-t;                                 // used time (in 1/100 secs)
↦ 12
size(sI);
↦ 17
t=timer;
sI=groebner(I);
↦ compute hilbert series with std in ring (ZZ/32003),(a,b,c,d,e,f,@),(dp(7)\
  ,C)
↦ weights used for hilbert series: 1,1,1,1,1,1,1
↦ [65535:2]1(5)s2(4)s3(3)s4ss5(4)s(5)s(7)-s6(8)s(9)s(11)s(13)s(16)s(18)s(21\
  )--s7(22)s(23)s(24)s(27)s(29)s(31)s(32)s(35)-s(37)s(40)s(42)s(44)s(45)--s\
  (46)s(48)-----8-s(44)s(47)s(50)s(52)s(55)s(57)s(59)s(61)-s(63)----s(62)--\
  --s(61)s(64)-s(66)-----------s(58)-------9-s(53)s(56)s(59)s(62)s(65)s(68)\
  s(71)s(74)s(77)s(80)s(83)s(86)s(90)s(95)s(102)s(108)--------(100)--------\
  --------------s(81)---10-s(83)s(88)s(90)s(94)s(99)s(104)s(109)s(114)-s(11\
  6)s(121)s(126)s(128)s(132)--------------------------------(100)----------\
  ----11-s(87)--------------------------------------12-s(50)--------13-s(4\
  4)s(47)s(51)s(55)-------------14-s(45)s(48)s(51)s(55)s(58)s(61)s(64)s(67)\
  s(70)--------------------15-s(52)s(55)s(58)s(61)s(64)s(67)s(70)s(73)s(76)\
  s(79)s(82)-------------------------------------16------------------------\
  --------------------17-
↦ product criterion:284 chain criterion:4184
↦ std with hilb in (ZZ/32003),(a,b,c,d,e,f,@),(lp(6),dp(1),C)
↦ [65535:2]1(98)s2(97)s3(96)s4s(97)-s5(98)s(101)s(103)s(106)s(109)---s6(107\
  )s(109)s(111)s(114)s(117)s(120)s(123)s(125)s(128)s(131)ss(132)-s---------\
  -s7(125)s(127)s(130)s(132)s(135)s(138)s(140)s(143)s(146)s(147)s(149)s(150\
  )s(152)s(154)s(157)s(159)s(162)---ss(165)-----------shhhhhhhhhhhhhhhhhhhh\
  hhh8(134)s(136)s(139)s(142)s(145)s(147)s(150)s(151)s(153)s(155)s(158)s(16\
  1)s(163)s(166)s(169)s(172)s(175)s(177)s(180)s(183)-----------------------\
  -s(171)s(178)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh9(147)s(150)s(153)s(155)s(\
  181)s(184)s(187)s(190)s(203)s(208)s(213)s(217)s(218)s(220)s(222)s(225)---\
  s-s(226)-----------s(219)---------shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh\
  hhhhhhhhhhh10(163)s(166)s(168)s(171)s(177)s(180)s(183)s(186)shhhhhhhhhhhh\
  hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh11(125)s(128)s(13\
  0)s(133)s(136)shhhhhhhhhhhhhhhhhhhhhhhhhhh12(110)s(113)s(120)s(123)s(12\
  7)-------------shhhhhhhhhhhhhh13(102)s(106)s(109)s(111)s(114)s(117)----s\
  hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh14(85)s(90)s(93)s(97)s(100)s(103)---(100)-\
  s(103)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh15(68)s(72)s(75)s(79)s(85)-\
  ---shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh16(42)s(45)s(49)shhhhhhhhh\
  hhhhhhhhhhhhh17(34)s(37)shhhhhhhhhhhhhh18(27)s(30)s(32)-shhhhhhhhh19(26)s\
  (29)s(32)shhhhhhhhhhhhh20(22)s(25)s(28)shhhhhhhhhhhhh21(20)s(26)shhhhhhhh\
  hhhh22(18)shhhhhhhhh23(12)shhhhh24(11)s(14)-shhhh25(13)s(18)-s(21)shhhhhh\
  h26(18)shhhhhhhhhhhh27(9)shhhhh28(8)shhhh29(7)shhhh30(8)-shhh31shhhhh32(7\
  )shhhh33shhhh34(6)shhhhhhhh36(2)s37(6)shhhh38shhhh39shhhhhhhh42(2)s43(5)s\
  hhhh44shhhhhhhh48s49shhhh50shhhhhhhh54shhhhh
↦ product criterion:720 chain criterion:11620
↦ hilbert series criterion:532
↦ dehomogenization
↦ simplification
↦ imap to ring (ZZ/32003),(a,b,c,d,e,f),(lp(6),C)
timer-t;                                 // used time (in 1/100 secs)
↦ 9
size(sI);
↦ 17
option(noprot);
```

A.2.2 Groebner basis conversion

The performance of Buchberger's algorithm is sensitive to the chosen monomial order. A Groebner basis computation with respect to a less favorable order such as the lexicographic ordering may easily run out of time or memory even in cases where a Groebner basis computation with respect to a more efficient order such as the degree reverse lexicographic ordering is very well feasible. Groebner basis conversion algorithms and the Hilbert-driven Buchberger algorithm are based on this observation:

- Groebner basis conversion: Given an ideal $I \subset K[x_1,\ldots,x_n]$ and a slow monomial order, compute a Groebner basis with respect to an appropriately chosen fast order. Then convert the result to a Groebner basis with respect to the given slow order.
- Hilbert-driven Buchberger algorithm: Homogenize the given generators for I with respect to a new variable, say, x_0 . Extend the given slow ordering on $K[x_1,\ldots,x_n]$ to a global product ordering on $K[x_0,\ldots,x_n]$. Compute a Groebner basis for the ideal generated by the homogenized polynomials with respect to a fast ordering. Read the Hilbert function, and use this information when computing a Groebner basis with respect to the extended (slow) ordering. Finally, dehomogenize the elements of the resulting Groebner basis.

Singular provides implementations for the FGLM conversion algorithm (which applies to zero-dimensional ideals only, see Section 5.1.152 [stdfglm], page 272) and variants of the Groebner walk conversion algorithm (which works for arbitrary ideals, See Section 5.1.49 [frwalk], page 189, Section D.4.10 [grwalk_lib], page 1087). An implementation of the Hilbert-driven Buchberger algorithm is accessible via the `stdhilb` command (see also Section 5.1.151 [std], page 271).

For the ideal below, `stdfglm` is more than 100 times and `stdhilb` about 10 times faster than `std`.

```
  ring r =32003,(a,b,c,d,e),lp;
  ideal i=a+b+c+d, ab+bc+cd+ae+de, abc+bcd+abe+ade+cde,
          abc+abce+abde+acde+bcde, abcde-1;
  int t=timer;
  option(prot);
  ideal j1=stdfglm(i);
↦ std in (ZZ/32003),(a,b,c,d,e),(dp(5),C)
↦ [1048575:3]1(4)s2(3)s3(2)s4s(3)s5s(4)s(5)s(6)6-ss(7)s(9)s(11)-7-ss(13)s(1\
   5)s(17)--s--8-s(16)s(18)s(20)s(23)s(26)-s(23)-------9--s(16)s10(19)s(22)s\
   (25)----s(24)--s11---------s12(17)s(19)s(21)------s(17)s(19)s(21)s13(23)s\
   --s-----s(20)----------14-s(12)--------15-s(6)--16-s(5)--17---
↦ (S:21)---------------------
↦ product criterion:109 chain criterion:322
↦ .....+...-...-..-+-...-...-..---...-++---++---...-...-++---.++-----------..---
↦ vdim= 45
↦ ............................................++----------------------------------
  timer-t;
↦ 0
  size(j1);   // size (no. of polys) in computed GB
↦ 5
  t=timer;
  ideal j2=stdhilb(i);
↦ compute hilbert series with std in ring (ZZ/32003),(a,b,c,d,e,@),(dp(6),C\
  )
↦ weights used for hilbert series: 1,1,1,1,1,1
↦ [1048575:2]1(4)s2(3)s3(2)s4ss5(3)s(4)s(5)-s6s(6)s(7)s(9)s(11)-7-ss(13)s(1\
  5)s(17)--s--8-s(16)s(18)s(20)s(23)s(26)-s(29)-------9-s(25)s(28)--s(29)--\
  -s-------10-s(24)-------s(19)---11-s(17)s(19)s(21)-----s(18)-s(19)s12(21)\
  s(23)s(26)-s(27)------s(23)----------13-s(15)-----------14-s(6)--15-s(5)-\
  -16---
↦ product criterion:88 chain criterion:650
↦ std with hilb in (ZZ/32003),(a,b,c,d,e,@),(lp(5),dp(1),C)
↦ [1048575:2]1(41)s2(40)s3(39)s4s(40)-s5(41)s(44)s(46)s-s-sh6s(49)s(51)s(54\
  )s(55)s(56)s(58)s(59)--shhhhhhh7(53)s(55)s(57)s(59)s(61)-s(62)s(68)s(70)s\
  (71)s(74)--shhhhhhhhhhhhhhhh8(58)s(61)s(65)s(68)s(71)-s(72)s(75)--------s\
  hhhhhhhhhhhhhhhhhhh9(51)s(53)s(56)s(58)s(61)s(64)------s(61)s(64)shhhhhhh\
  hhhhhhhh10(53)s(55)s(58)s(62)s(64)s(67)s(70)--s(71)------s(68)s(71)s(73)-\
  -shhhhhhhhhhhhhhhh11(58)s(60)s(63)s(66)s(69)s(72)s(74)---s-s(76)s(79)----\
  s(78)-------shhhhhhhhhhhhhhhhhhhhhhh12(51)s(54)s(57)s(58)s(60)s(63)s(65)s\
  (68)s(70)s(73)s(76)s(79)--s(80)----shhhhhhhhhhhhhhhhhhhhhhhhhhhhhh13(48)s\
  (51)s(54)s(57)s(59)s(61)s(64)s(67)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh\
  h14(31)s(33)s(36)s(39)s(42)s(45)shhhhhhhhhhhhhhhhhhhhhhhhh15(23)s(26)s(29\
  )s(32)s(35)shhhhhhhhhhhhhhhhhhh16(18)s(21)s(24)s(27)shhhhhhhhhhhhhhh17(15\
  )s(18)s(21)s(24)shhhhhhhhhhhh18(15)s(18)s(21)s(24)shhhhhhhhhhhh19(14)s(17\
  )s(20)shhhhhhhhhhhh20(11)s(14)s(17)shhhhhhhhh21(11)s(14)s(17)shhhhhhhhh22\
  (11)s(14)s(16)shhhhhhhhh23(10)s(13)shhhhhhhhh24(7)s(10)shhhhhh25(7)s(10)s\
  hhhhhh26(7)s(10)shhhhhh27(7)s(10)shhhhhh28(7)s(10)shhhhhh29(7)s(10)shhhhh\
  h30(7)s(9)shhhhhh31(6)shhhhhh32(3)shhh33shhh34shhh35shhh36shhh37shhh38shh\
  h39shhh40shhh41shhh42shhh43shhh44shhh45shhh46shhh47shhh48shhh49shhh50shhh\
  51shhh52shhh53shhh54shhhhhh
↦ product criterion:491 chain criterion:11799
↦ hilbert series criterion:417
↦ dehomogenization
↦ simplification
↦ imap to ring (ZZ/32003),(a,b,c,d,e),(lp(5),C)
  timer-t;
↦ 0
  size(j2);   // size (no. of polys) in computed GB
↦ 5
  // usual Groebner basis computation for lex ordering
  t=timer;
  ideal j0 =std(i);
↦ [1048575:2]1(4)s2(3)s3(2)s4s(3)s5(5)s(4)s6(6)s(7)s(9)s(8)sss7(10)s(11)s(1\
  0)s(11)s(13)s8(12)s(13)s(15)s.s(14).s.9.s(16)s(17)s(19)........10.s(20).s\
  (21)ss..11.s(23)s(25).ss(27)...s(28)s(26)...12.s(25)sss(23)sss.......s(22\
  )...13.s(23)ssssssss(21)s(22)sssss(21)ss..14.ss(22)s.s.sssss(21)s(22)sss.\
  s...15.ssss(21)s(22)ssssssssss(21)s(22)sss16.ssssssss(21)s(22)sssssssssss\
  17ss(21)s(22)ssssssssss(21)sss(22)ss(21)ss18(22)s(21)s(22)s.s............\
  ..19.sssss(21)ss(22)ssssssssss(21)s(22)s20.ssssssssss(21)s........21.s(22\
  )sssssssssssssss(21)s(22)ssss22ssssssssssss(21)s(22)sssssss23sssssssssss(\
  21)s(22)ssssssss24ssssssssssss(21)s(22)sssssss25ssssssssss(21)s(22)ssssss\
  sss26ssssssssss(21)s(20)ssssssss27.sssssssss..........s28.ssssss.........\
  ....29.sssssssssssssssssss30sssssssssssssssssss31.sssssssssssssssssss32.s\
  sssssssssssssssssss33ssssssssssssssssssss34ssssssssssssssssssss35ssssssss\
  ssssssssssss36ssssssssssssssssssss37ssssssssssssssssssss38sssssssssssssss\
  sssss39ssssssssssssssssssss40ssssssssssssssssssss41ssss---------------42-\
  s(4)--43-s44s45s46s47s48s49s50s51s52s53s54s55s56s
↦ product criterion:1395 chain criterion:904
  option(noprot);
  timer-t;
↦ 0
```

A.2.3 slim Groebner bases

The command `slimgb` calls an implementation of an algorithm to compute Groebner bases which is designed for keeping the polynomials slim (short with small coefficients) during a Groebner basis computation. It provides, in particular, a fast algorithm for computing Groebner bases over function fields or over the rational numbers, but also in several other cases. The algorithm which is still under development was developed in the diploma thesis of Michael Brickenstein. It has been published as `https://www.singular.uni-kl.de/reports/35/paper_35_full.ps.gz`.

In the example below (Groebner basis with respect to degree reverse lexicographic ordering over function field) `slimgb` is much faster than the `std` command.

```
ring r=(32003,u1, u2, u3, u4),(x1, x2, x3, x4, x5, x6, x7),dp;
timer=1;
ideal i=
-x4*u3+x5*u2,
x1*u3+2*x2*u1-2*x2*u2-2*x3*u3-u1*u4+u2*u4,
-2*x1*x5+4*x4*x6+4*x5*x7+x1*u3-2*x4*u1-2*x4*u4-2*x6*u2-2*x7*u3+u1*u2+u2*u4,
-x1*x5+x1*x7-x4*u1+x4*u2-x4*u4+x5*u3+x6*u1-x6*u2+x6*u4-x7*u3,
-x1*x4+x1*u1-x5*u1+x5*u4,
-2*x1*x3+x1*u3-2*x2*u4+u1*u4+u2*u4,
x1^2*u3+x1*u1*u2-x1*u2^2-x1*u3^2-u1*u3*u4+u3*u4^2;
i=slimgb(i);
```

For detailed information and limitations see Section 5.1.145 [slimgb], page 265.

A.3 Commutative Algebra

A.3.1 Saturation

For any two ideals i, j in the basering R let

$$\operatorname{sat}(i,j) = \{x \in R \mid \exists\, n \text{ s.t. } x \cdot (j^n) \subseteq i\} = \bigcup_{n=1}^{\infty} i : j^n$$

denote the saturation of i with respect to j . This defines, geometrically, the closure of the complement of V(j) in V(i) (where V(i) denotes the variety defined by i).

The saturation is computed by the procedure `sat` in `elim.lib` by computing iterated ideal quotients with the maximal ideal. `sat` returns a list of two elements: the saturated ideal and the number of iterations.

We apply saturation to show that a variety has no singular points outside the origin (see also Section A.4.2 [Critical points], page 736). We choose m to be the homogeneous maximal ideal (note that `maxideal(n)` denotes the n-th power of the maximal ideal). Then $V(i)$ has no singular point outside the origin if and only if $sat(j+(f),m)$ is the whole ring, that is, generated by 1.

```
    LIB "elim.lib";         // loading library elim.lib
    ring r2 = 32003,(x,y,z),dp;
    poly f = x^11+y^5+z^(3*3)+x^(3+2)*y^(3-1)+x^(3-1)*y^(3-1)*z3+
      x^(3-2)*y^3*(y^2)^2;
    ideal j=jacob(f);
    sat(j+f,maxideal(1));
  ↦ _[1]=1
    // list the variables defined so far:
    listvar();
  ↦ // r2                             [0]  *ring
  ↦ //      j                              [0]  ideal, 3 generator(s)
  ↦ //      f                              [0]  poly
```

A.3.2 Finite fields

We define a variety in the n -space of codimension 2 defined by polynomials of degree d with generic coefficients over the prime field Z/p and look for zeros on the torus. First over the prime field and then in the finite extension field with p^k elements. In general there will be many more solutions in the second case. (Since the SINGULAR language is interpreted, the evaluation of many `for`-loops is not very fast):

```
    int p=3;  int n=3;  int d=5; int k=2;
    ring rp = p,(x(1..n)),dp;
    int s = size(maxideal(d));
    s;
  ↦ 21
    // create a dense homogeneous ideal m, all generators of degree d, with
    // generic (random) coefficients:
    ideal m = maxideal(d)*random(p,s,n-2);
    m;
  ↦ m[1]=x(1)^3*x(2)^2-x(1)*x(2)^4+x(1)^4*x(3)-x(1)^3*x(2)*x(3)+x(1)*x(2)^3*x\
     (3)+x(2)^4*x(3)+x(2)^3*x(3)^2+x(1)*x(2)*x(3)^3+x(1)*x(3)^4-x(3)^5
    // look for zeros on the torus by checking all points (with no component 0)
    // of the affine n-space over the field with p elements :
    ideal mt;
    int i(1..n);                    // initialize integers i(1),...,i(n)
    int l;
    s=0;
    for (i(1)=1;i(1)<p;i(1)=i(1)+1)
    {
      for (i(2)=1;i(2)<p;i(2)=i(2)+1)
      {
        for (i(3)=1;i(3)<p;i(3)=i(3)+1)
        {
          mt=m;
          for (l=1;l<=n;l=l+1)
     {
      mt=subst(mt,x(l),i(l));
     }
     if (size(mt)==0)
     {
      "solution:",i(1..n);
      s=s+1;
     }
    }
   }
  }
↦ solution: 1 1 2
↦ solution: 1 2 1
↦ solution: 1 2 2
↦ solution: 2 1 1
↦ solution: 2 1 2
↦ solution: 2 2 1
  "//",s,"solutions over GF("+string(p)+")";
↦ // 6 solutions over GF(3)
  // Now go to the field with p^3 elements:
  // As long as there is no map from Z/p to the field with p^3 elements
  // implemented, use the following trick: convert the ideal to be mapped
  // to the new ring to a string and then execute this string in the
  // new ring
  string ms="ideal m="+string(m)+";";
  ms;
↦ ideal m=x(1)^3*x(2)^2-x(1)*x(2)^4+x(1)^4*x(3)-x(1)^3*x(2)*x(3)+x(1)*x(2)^\
  3*x(3)+x(2)^4*x(3)+x(2)^3*x(3)^2+x(1)*x(2)*x(3)^3+x(1)*x(3)^4-x(3)^5;
  // define a ring rpk with p^k elements, call the primitive element z. Hence
  // 'solution exponent: 0 1 5' means that (z^0,z^1,z^5) is a solution
  ring rpk=(p^k,z),(x(1..n)),dp;
  rpk;
↦ // coefficients: ZZ/9[z]
↦ //   minpoly        : 1*z^2+2*z^1+2*z^0
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
  execute(ms);
  s=0;
  ideal mt;
  for (i(1)=0;i(1)<p^k-1;i(1)=i(1)+1)
  {
   for (i(2)=0;i(2)<p^k-1;i(2)=i(2)+1)
   {
    for (i(3)=0;i(3)<p^k-1;i(3)=i(3)+1)
    {
     mt=m;
     for (l=1;l<=n;l=l+1)
     {
      mt=subst(mt,x(l),z^i(l));
     }
     if (size(mt)==0)
```

```
          {
            // we show only the first 7 solutions here:
            if (s<5) {"solution exponent:",i(1..n);}
            s=s+1;
          }
        }
      }
    }
  ↦ solution exponent: 0 0 2
  ↦ solution exponent: 0 0 4
  ↦ solution exponent: 0 0 6
  ↦ solution exponent: 0 4 0
  ↦ solution exponent: 0 4 1
    "//",s,"solutions over GF("+string(p^k)+")";
  ↦ // 72 solutions over GF(9)
```

A.3.3 Elimination

Elimination is the algebraic counterpart of the geometric concept of projection. If $f = (f_1, \ldots, f_n) : k^r \to k^n$ is a polynomial map, the Zariski-closure of the image is the zero-set of the ideal $j = J \cap k[x_1, \ldots, x_n]$, where

$$J = (x_1 - f_1(t_1, \ldots, t_r), \ldots, x_n - f_n(t_1, \ldots, t_r)) \subseteq k[t_1, \ldots, t_r, x_1, \ldots, x_n]$$

that is, of the ideal j obtained from J by eliminating the variables $t_1, \ldots, t_r$. This can be done by computing a Groebner basis for J with respect to a (global) product ordering where the block of t-variables precedes the block of x-variables, and then selecting those polynomials which do not contain any t. Alternatively, we may use a global monomial ordering with extra weight vector (see Section B.2.8 [Extra weight vector], page 772), assigning to the t-variables a positive weight and to the x-variables weight 0.

Since elimination is expensive, it may be useful to use a Hilbert-driven approach to the elimination problem (see Section A.2.2 [Groebner basis conversion], page 712):

First compute the Hilbert function of the ideal w.r.t. a fast ordering (e.g., `dp`), then make use of it to speed up the computation by a Hilbert-driven elimination which uses the `intvec` provided as third argument.

In Singular the most convenient way is to use the `eliminate` command. In contrast to the first method, with `eliminate` the result needs not be a standard basis in the given ordering. Hence, there may be cases where the first method is the preferred one.

WARNING: In the case of a local or a mixed ordering, elimination needs special care. f may be considered as a map of germs $f : (k^r, 0) \to (k^n, 0)$, but even if this map germ is finite, we are in general not able to compute the image germ because for this we would need an implementation of the Weierstrass preparation theorem. What we can compute, and what `eliminate` actually does, is the following: let V(J) be the zero-set of J in $k^r \times (k^n, 0)$, then the closure of the image of V(J) under the projection

$$\mathrm{pr} : k^r \times (k^n, 0) \to (k^n, 0)$$

can be computed. (Note that this germ contains also those components of V(J) which meet the fiber of pr outside the origin.) This is achieved by an ordering with the block of t-variables having a global ordering (and preceding the x-variables) and the x-variables having a local ordering.

In any case, if the input is weighted homogeneous (=quasihomogeneous), the weights given to the variables should be chosen accordingly. Singular offers a function `weight` which proposes, given an ideal or module, integer weights for the variables, such that the ideal, resp. module, is as homogeneous as possible with respect to these weights. The function finds correct weights, if the input is weighted homogeneous (but is rather slow for many variables). In order to check, whether the input is quasihomogeneous, use the function `qhweight`, which returns an `intvec` of correct weights if the input is quasihomogeneous and an `intvec` of zeros otherwise.

Let us give three examples:

1. First we compute the equations of the simple space curve ’T[7]’ consisting of two tangential cusps given in parametric form.
2. We compute weights for the equations such that the equations are quasihomogeneous w.r.t. these weights.
3. Then we compute the tangent developable of the rational normal curve in P^4.

```
  // 1. Compute equations of curve given in parametric form:
  // Two transversal cusps in (k^3,0):
  ring r1 = 0,(t,x,y,z),ls;
  ideal i1 = x-t2,y-t3,z;        // parametrization of the first branch
  ideal i2 = y-t2,z-t3,x;        // parametrization of the second branch
  ideal j1 = eliminate(i1,t);
  j1;                            // equations of the first branch
↦ j1[1]=z
↦ j1[2]=y2-x3
  ideal j2 = eliminate(i2,t);
  j2;                            // equations of the second branch
↦ j2[1]=z2-y3
↦ j2[2]=x
  // Now map to a ring with only x,y,z as variables and compute the
  // intersection of j1 and j2 there:
  ring r2 = 0,(x,y,z),ds;
  ideal j1= imap(r1,j1);         // imap is a convenient ringmap for
  ideal j2= imap(r1,j2);         // inclusions and projections of rings
  ideal i = intersect(j1,j2);
  i;                             // equations of both branches
↦ i[1]=z2-y3+x3y
↦ i[2]=xz
↦ i[3]=xy2-x4
  //
  // 2. Compute the weights:
  intvec v= qhweight(i);         // compute weights
  v;
↦ 4,6,9
  //
  // 3. Compute the tangent developable
  // The tangent developable of a projective variety given parametrically
  // by F=(f1,...,fn) : P^r --> P^n is the union of all tangent spaces
  // of the image. The tangent space at a smooth point F(t1,...,tr)
  // is given as the image of the tangent space at (t1,...,tr) under
  // the tangent map (affine coordinates)
  //   T(t1,...,tr): (y1,...,yr) --> jacob(f)*transpose((y1,...,yr))
  // where jacob(f) denotes the jacobian matrix of f with respect to the
  // t’s evaluated at the point (t1,...,tr).
  // Hence we have to create the graph of this map and then to eliminate
  // the t’s and y’s.
  // The rational normal curve in P^4 is given as the image of
  //        F(s,t) = (s4,s3t,s2t2,st3,t4)
  // each component being homogeneous of degree 4.
  ring P = 0,(s,t,x,y,a,b,c,d,e),dp;
  ideal M = maxideal(1);
  ideal F = M[1..2];     // take the 1st two generators of M
  F=F^4;
  // simplify(...,2); deletes 0-columns
  matrix jac = simplify(jacob(F),2);
  ideal T = x,y;
  ideal J = jac*transpose(T);
  ideal H = M[5..9];
  ideal i = matrix(H)-matrix(J);// this is tricky: difference between two
                         // ideals is not defined, but between two
                         // matrices. By type conversion
                         // the ideals are converted to matrices,
                         // subtracted and afterwards converted
                         // to an ideal. Note that '+' is defined
                         // and adds (concatenates) two ideals
  i;
↦ i[1]=-4s3x+a
↦ i[2]=-3s2tx-s3y+b
↦ i[3]=-2st2x-2s2ty+c
↦ i[4]=-t3x-3st2y+d
↦ i[5]=-4t3y+e
  // Now we define a ring with product ordering and weights 4
  // for the variables a,...,e.
  // Then we map i from P to P1 and eliminate s,t,x,y from i.
  ring P1 = 0,(s,t,x,y,a,b,c,d,e),(dp(4),wp(4,4,4,4,4));
  ideal i = fetch(P,i);
  ideal j= eliminate(i,stxy);    // equations of tangent developable
  j;
↦ j[1]=3c2-4bd+ae
↦ j[2]=2bcd-3ad2-3b2e+4ace
↦ j[3]=8b2d2-9acd2-9b2ce+14abde-4a2e2
  // We can use the product ordering to eliminate s,t,x,y from i
  // by a std-basis computation.
  // We need proc 'nselect' from elim.lib.
  LIB "elim.lib";
  j = std(i);                    // compute a std basis j
  j = nselect(j,1..4);           // select generators from j not
  j;                             // containing variable 1,...,4
↦ j[1]=3c2-4bd+ae
↦ j[2]=2bcd-3ad2-3b2e+4ace
↦ j[3]=8b2d2-9acd2-9b2ce+12ac2e-2abde
```

A.3.4 Free resolution

In Singular a free resolution of a module or ideal has its own type: `resolution`. It is a structure that stores all information related to free resolutions. This allows partial computations of resolutions via the command `res`. After applying `res`, only a pre-format of the resolution is computed which allows to determine invariants like Betti-numbers or homological dimension. To see the differentials of the complex, a resolution must be converted into the type list which yields a list of modules:

the k-th module in this list is the first syzygy-module (module of relations) of the (k-1)st module. There are the following commands to compute a resolution:

`res` Section 5.1.134 [res], page 253
computes a free resolution of an ideal or module using a heuristically chosen method. This is the preferred method to compute free resolutions of ideals or modules.

`fres` Section 5.1.48 [fres], page 188
improved version of Section 5.1.149 [sres], page 269, computes a free resolution of an ideal or module using Schreyer's method. The input has to be a standard basis.

`lres` Section 5.1.83 [lres], page 215
computes a free resolution of an ideal or module with LaScala's method. The input needs to be homogeneous.

`mres` Section 5.1.98 [mres], page 225
computes a minimal free resolution of an ideal or module with the syzygy method.

`sres` Section 5.1.149 [sres], page 269
computes a free resolution of an ideal or module with Schreyer's method. The input has to be a standard basis.

`nres` Section 5.1.106 [nres], page 232
computes a free resolution of an ideal or module with the standard basis method.

`minres` Section 5.1.93 [minres], page 223
minimizes a free resolution of an ideal or module.

`syz` Section 5.1.156 [syz], page 280
computes the first syzygy module.

`res(i,r)`, `lres(i,r)`, `sres(i,r)`, `mres(i,r)`, `nres(i,r)` compute the first r modules of the resolution of i, resp. the full resolution if r=0 and the basering is not a qring. See the manual for a precise description of these commands.
Note: The command `betti` does not require a minimal resolution for the minimal Betti numbers.

Now let us take a look at an example which uses resolutions: The Hilbert-Burch theorem says that the ideal i of a reduced curve in K^3 has a free resolution of length 2 and that i is given by the 2x2 minors of the 2nd matrix in the resolution. We test this for two transversal cusps in K^3. Afterwards we compute the resolution of the ideal j of the tangent developable of the rational normal curve in P^4 from above. Finally we demonstrate the use of the type `resolution` in connection with the `lres` command.

```
  // Two transversal cusps in (k^3,0):
  ring r2 =0,(x,y,z),ds;
  ideal i =z2-1y3+x3y,xz,-1xy2+x4,x3z;
  resolution rs=mres(i,0);   // computes a minimal resolution
  rs;                        // the standard representation of complexes
↦   1       3       2
↦ r2 <--  r2 <--  r2
↦ 
↦ 0       1       2
↦ 
    list resi=rs;            // conversion to a list
  print(resi[1]);            // the 1st module is i minimized
↦ xz,
↦ z2-y3+x3y,
↦ xy2-x4
```

```
  print(resi[2]);            // the 1st syzygy module of i
↦ -z,-y2+x3,
↦ x, 0,
↦ y, z
  resi[3];                   // the 2nd syzygy module of i
↦ _[1]=0
  ideal j=minor(resi[2],2);
  reduce(j,std(i));          // check whether j is contained in i
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
  size(reduce(i,std(j)));    // check whether i is contained in j
↦ 0
  // size(<ideal>) counts the non-zero generators
  // -------------------------------------------
  // The tangent developable of the rational normal curve in P^4:
  ring P = 0,(a,b,c,d,e),dp;
  ideal j= 3c2-4bd+ae, -2bcd+3ad2+3b2e-4ace,
           8b2d2-9acd2-9b2ce+9ac2e+2abde-1a2e2;
  resolution rs=mres(j,0);
  rs;
↦  1      2      1
↦ P <--  P <--  P
↦
↦ 0      1      2
↦
  list L=rs;
  print(L[2]);
↦ 2bcd-3ad2-3b2e+4ace,
↦ -3c2+4bd-ae
  // create an intmat with graded Betti numbers
  intmat B=betti(rs);
  // this gives a nice output of Betti numbers
  print(B,"betti");
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     1     -
↦     2:     -     1     -
↦     3:     -     -     1
↦ ------------------------
↦ total:     1     2     1
↦
  // the user has access to all Betti numbers
  // the 2-nd column of B:
  B[1..4,2];
↦ 0 1 1 0
  ring cyc5=32003,(a,b,c,d,e,h),dp;
  ideal i=
  a+b+c+d+e,
  ab+bc+cd+de+ea,
  abc+bcd+cde+dea+eab,
  abcd+bcde+cdea+deab+eabc,
  h5-abcde;
  resolution rs=lres(i,0);   //computes the resolution according LaScala
  rs;                        //the shape of the minimal resolution
↦     1         5         10          10         5         1
↦ cyc5 <--  cyc5 <--  cyc5 <--   cyc5 <--   cyc5 <--  cyc5
↦
↦ 0         1         2          3          4         5
↦
  print(betti(rs),"betti");  //shows the Betti-numbers of cyclic 5
↦            0     1     2     3     4     5
↦ ------------------------------------------
↦     0:     1     1     -     -     -     -
↦     1:     -     1     1     -     -     -
↦     2:     -     1     1     -     -     -
↦     3:     -     1     2     1     -     -
↦     4:     -     1     2     1     -     -
↦     5:     -     -     2     2     -     -
↦     6:     -     -     1     2     1     -
↦     7:     -     -     1     2     1     -
↦     8:     -     -     -     1     1     -
↦     9:     -     -     -     1     1     -
↦    10:     -     -     -     -     1     1
↦ ------------------------------------------
↦ total:     1     5    10    10     5     1
↦
  dim(rs);                   //the homological dimension
↦ 4
  size(list(rs));            //gets the full (non-reduced) resolution
↦ 6
  minres(rs);                //minimizes the resolution
↦     1         5         10          10         5         1
↦ cyc5 <--  cyc5 <--  cyc5 <--   cyc5 <--   cyc5 <--  cyc5
↦
↦ 0         1         2          3          4         5
↦
  size(list(rs));            //gets the minimized resolution
↦ 6
```

A.3.5 Handling graded modules

How to deal with graded modules in Singular is best explained by looking at an example:

```
ring R = 0, (w,x,y,z), dp;
module I = [-x,0,-z2,0,y2z], [0,-x,-yz,0,y3], [-w,0,0,yz,-z3],
           [0,-w,0,y2,-yz2], [0,-1,-w,0,xz], [0,-w,0,y2,-yz2],
           [x2,-y2,-wy2+xz2];
print(I);
↦ -x, 0,  -w, 0,   0, 0,   x2,
↦ 0,  -x, 0,  -w,  -1,-w,  -y2,
↦ -z2,-yz,0,  0,   -w,0,   -wy2+xz2,
↦ 0,  0,  yz, y2,  0, y2,  0,
↦ y2z,y3, -z3,-yz2,xz,-yz2,0

// (1) Check on degrees:
```

```
// ====================
attrib(I,"isHomog"); // attribute not set => empty output
↦
homog(I);
↦ 1
attrib(I,"isHomog");
↦ 2,2,1,1,0

print(betti(I,0),"betti");  // read degrees from Betti diagram
↦            0     1
↦ ------------------
↦      0:     1     -
↦      1:     2     1
↦      2:     2     5
↦      3:     -     1
↦ ------------------
↦ total:     5     7
↦

// (2) Shift degrees:
// ==================
def J=I;
intvec DV = 0,0,-1,-1,-2;
attrib(J,"isHomog",DV);    // assign new weight vector
attrib(J,"isHomog");
↦ 0,0,-1,-1,-2
print(betti(J,0),"betti");
↦            0     1
↦ ------------------
↦     -2:     1     -
↦     -1:     2     1
↦      0:     2     5
↦      1:     -     1
↦ ------------------
↦ total:     5     7
↦

intmat bettiI=betti(I,0);  // degree corresponding to first non-zero row
                           // of Betti diagram is accessible via
                           // attribute "rowShift"
attrib(bettiI);
↦ attr:rowShift, type int
intmat bettiJ=betti(J,0);
attrib(bettiJ);
↦ attr:rowShift, type int

// (3) Graded free resolutions:
// ============================
resolution resJ = mres(J,0);
attrib(resJ);
↦ attr:isHomog, type intvec
print(betti(resJ),"betti");
↦            0     1     2
↦ ------------------------
↦    -2:      1     -     -
↦    -1:      2     -     -
↦     0:      1     4     -
↦     1:      -     -     1
↦ ------------------------
↦ total:      4     4     1
↦
attrib(betti(resJ));
↦ attr:rowShift, type int
```

A check on degrees ((1), by using the `homog` command) shows that this is a graded matrix. The `homog` command assigns an admissible weight vector (here: 2,2,1,1,0) to the module `I` which is accessible via the attribute `"isHomog"`. Thus, we may think of `I` as a graded submodule of the graded free R -module

$$F = R(-2)^2 \oplus R(-1)^2 \oplus R\,.$$

We may also read the degrees from the Betti diagram as shown above. The degree on the left of the first nonzero row of the Betti diagram is accessible via the attribute `"rowShift"` of the betti matrix (which is of type `intmat`):

(2) We may shift degrees by assigning another admissible degree vector. Note that Singular does not check whether the assigned degree vector really is admissible. Moreover, note that all assigned attributes are lost under a type conversion (e.g. from `module` to `matrix`).

(3) These considerations may be applied when computing data from free resolutions (see Section A.3.6 [Computation of Ext], page 724).

A.3.6 Computation of Ext

We start by showing how to calculate the n-th Ext group of an ideal. The ingredients to do this are by the definition of Ext the following: calculate a (minimal) resolution at least up to length n, apply the Hom functor, and calculate the n-th homology group, that is, form the quotient ker/im in the resolution sequence.

The Hom functor is given simply by transposing (hence dualizing) the module or the corresponding matrix with the command `transpose`. The image of the (n-1)-st map is generated by the columns of the corresponding matrix. To calculate the kernel apply the command `syz` at the (n-1)-st transposed entry of the resolution. Finally, the quotient is obtained by the command `modulo`, which gives for two modules A = ker, B = Im the module of relations of

$$A/(A \cap B)$$

in the usual way. As we have a chain complex, this is obviously the same as ker/Im.

We collect these statements in the following short procedure:

```
proc ext(int n, ideal I)
{
  resolution rs = mres(I,n+1);
  module tAn    = transpose(rs[n+1]);
  module tAn_1  = transpose(rs[n]);
  module ext_n  = modulo(syz(tAn),tAn_1);
  return(ext_n);
}
```

Now consider the following example:

```
ring r5 = 32003,(a,b,c,d,e),dp;
ideal I = a2b2+ab2c+b2cd, a2c2+ac2d+c2de,a2d2+ad2e+bd2e,a2e2+abe2+bce2;
print(ext(2,I));
↦ 1,0,0,0,0,0,0,
↦ 0,1,0,0,0,0,0,
↦ 0,0,1,0,0,0,0,
↦ 0,0,0,1,0,0,0,
↦ 0,0,0,0,1,0,0,
↦ 0,0,0,0,0,1,0,
↦ 0,0,0,0,0,0,1
ext(3,I);   // too big to be displayed here
```

The library `homolog.lib` contains several procedures for computing Ext-modules and related modules, which are much more general and sophisticated than the above one. They are used in the following example:

If M is a module, then $\mathrm{Ext}^1(M,M)$, resp. $\mathrm{Ext}^2(M,M)$, are the modules of infinitesimal deformations, respectively of obstructions, of M (like T1 and T2 for a singularity). Similar to the treatment of singularities, the semiuniversal deformation of M can be computed (if Ext^1 is finite dimensional) with the help of Ext^1, Ext^2 and the cup product. There is an extra procedure for $\mathrm{Ext}^k(R/J,R)$ if J is an ideal in R , since this is faster than the general Ext.

We compute

- the infinitesimal deformations ($= \mathrm{Ext}^1(K,K)$) and obstructions ($= \mathrm{Ext}^2(K,K)$) of the residue field $K = R/m$ of an ordinary cusp, $R = K[x,y]_m/(x^2-y^3)$, $m = (x,y)$. To compute $\mathrm{Ext}^1(m,m)$ we have to apply `Ext(1,syz(m),syz(m))` with `syz(m)` the first syzygy module of m , which is isomorphic to $\mathrm{Ext}^2(K,K)$.
- $\mathrm{Ext}^k(R/i,R)$ for some ideal i and with an extra option.

```
  LIB "homolog.lib";
  ring R=0,(x,y),ds;
  ideal i=x2-y3;
  qring q = std(i);      // defines the quotient ring k[x,y]_m/(x2-y3)
  ideal m = maxideal(1);
  module T1K = Ext(1,m,m);  // computes Ext^1(R/m,R/m)
↦ // dimension of Ext^1:  0
↦ // vdim of Ext^1:       2
↦
  print(T1K);
↦ 0,x,0,y,
↦ x,0,y,0
  printlevel=2;             // gives more explanation
  module T2K=Ext(2,m,m);    // computes Ext^2(R/m,R/m)
↦ // Computing Ext^2 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F2,G0)-->Hom(F3,G0) is given by:
↦ y2,x,
↦ x, y
↦ // and Hom(F1,G0) + Hom(F2,G1)-->Hom(F2,G0) is given by:
↦ -y,x,  x,0,y,0,
↦ x, -y2,0,x,0,y
↦
↦ // dimension of Ext^2:  0
↦ // vdim of Ext^2:       2
```

```
↦
  print(std(T2K));
↦ 0,x,0,y,
↦ x,0,y,0
  printlevel=0;
  module E = Ext(1,syz(m),syz(m));
↦ // dimension of Ext^1:  0
↦ // vdim of Ext^1:       2
↦
  print(std(E));
↦ 0,0,0,x,0,y,
↦ 0,0,x,0,y,0,
↦ 0,1,0,0,0,0,
↦ 1,0,0,0,0,0
  //The matrices which we have just computed are presentation matrices
  //of the modules T2K and E. Hence we may ignore those columns
  //containing 1 as an entry and see that T2K and E are isomorphic
  //as expected, but differently presented.
  //-------------------------------------------
  ring S=0,(x,y,z),dp;
  ideal  i = x2y,y2z,z3x;
  module E = Ext_R(2,i);
↦ // dimension of Ext^2:  1
↦
  print(E);
↦ 0,y,0,z2,
↦ z,0,0,-x,
↦ 0,0,x,-y
  // if a 3-rd argument of type int is given,
  // a list of Ext^k(R/i,R), a SB of Ext^k(R/i,R) and a vector space basis
  // is returned:
  list LE = Ext_R(3,i,0);
↦ // dimension of Ext^3:  0
↦ // vdim of Ext^3:       2
↦
  LE;
↦ [1]:
↦    _[1]=y*gen(1)
↦    _[2]=x*gen(1)
↦    _[3]=z2*gen(1)
↦ [2]:
↦    _[1]=y*gen(1)
↦    _[2]=x*gen(1)
↦    _[3]=z2*gen(1)
↦ [3]:
↦    _[1,1]=z
↦    _[1,2]=1
  print(LE[2]);
↦ y,x,z2
  print(kbase(LE[2]));
↦ z,1
```

A.3.7 Depth

We compute the depth of the module of Kaehler differentials $\mathrm{D}_k(\mathrm{R})$ of the variety defined by the $(m+1)$ -minors of a generic symmetric $(n \times n)$-matrix. We do this by computing the resolution over the polynomial ring. Then, by the Auslander-Buchsbaum formula, the depth is equal to the number of variables minus the length of a minimal resolution. This example was suggested by U. Vetter in order to check whether his bound $\mathrm{depth}(\mathrm{D}_k(R)) \geq m(m+1)/2+m-1$ could be improved.

```
  LIB "matrix.lib"; LIB "sing.lib";
  int n = 4;
  int m = 3;
  int N = n*(n+1) div 2;       // will become number of variables
  ring R = 32003,x(1..N),dp;
  matrix X = symmat(n);        // proc from matrix.lib
                               // creates the symmetric generic nxn matrix
  print(X);
↦ x(1),x(2),x(3),x(4),
↦ x(2),x(5),x(6),x(7),
↦ x(3),x(6),x(8),x(9),
↦ x(4),x(7),x(9),x(10)
  ideal J = minor(X,m);
  J=std(J);
  // Kaehler differentials D_k(R)
  // of R=k[x1..xn]/J:
  module D = J*freemodule(N)+transpose(jacob(J));
  ncols(D);
↦ 110
  nrows(D);
↦ 10
  //
  // Note: D is a submodule with 110 generators of a free module
  // of rank 10 over a polynomial ring in 10 variables.
  // Compute a full resolution of D with sres.
  // This takes about 17 sec on a Mac PB 520c and 2 sec an a HP 735
  int time = timer;
  module sD = std(D);
  list Dres = sres(sD,0);                // the full resolution
  timer-time;                            // time used for std + sres
↦ 0
  intmat B = betti(Dres);
  print(B,"betti");
↦            0     1     2     3     4     5     6
↦ ------------------------------------------------
↦     0:    10     -     -     -     -     -     -
↦     1:     -    10     -     -     -     -     -
↦     2:     -    84   144    60     -     -     -
↦     3:     -     -    35    80    60    16     1
↦ ------------------------------------------------
↦ total:    10    94   179   140    60    16     1
↦
  N-ncols(B)+1;                          // the desired depth
↦ 4
```

A.3.8 Factorization

The factorization of polynomials is implemented in the C++ libraries Factory (written mainly by Ruediger Stobbe) and libfac (written by Michael Messollen) which are part of the Singular system. For the factorization of univariate polynomials these libraries make use of the library NTL written by Victor Shoup.

```
  ring r = 0,(x,y),dp;
  poly f = 9x16-18x13y2-9x12y3+9x10y4-18x11y2+36x8y4
         +18x7y5-18x5y6+9x6y4-18x3y6-9x2y7+9y8;
  // = 9 * (x5-1y2)^2 * (x6-2x3y2-1x2y3+y4)
  factorize(f);
↦ [1]:
↦    _[1]=9
↦    _[2]=x6-2x3y2-x2y3+y4
↦    _[3]=-x5+y2
↦ [2]:
↦    1,1,2
  // returns factors and multiplicities,
  // first factor is a constant.
  poly g = (y4+x8)*(x2+y2);
  factorize(g);
↦ [1]:
↦    _[1]=1
↦    _[2]=x2+y2
↦    _[3]=x8+y4
↦ [2]:
↦    1,1,1
  // The same in characteristic 2:
  ring s = 2,(x,y),dp;
  poly g = (y4+x8)*(x2+y2);
  factorize(g);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+y
↦    _[3]=x2+y
↦ [2]:
↦    1,2,4
  // factorization over algebraic extension fields
  ring rext = (0,i),(x,y),dp;
  minpoly = i2+1;
  poly g = (y4+x8)*(x2+y2);
  factorize(g);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+(i)*y
↦    _[3]=x+(-i)*y
↦    _[4]=x4+(i)*y2
↦    _[5]=x4+(-i)*y2
↦ [2]:
↦    1,1,1,1,1
```

A.3.9 Primary decomposition

There are two algorithms implemented in Singular which provide primary decomposition: `primdecGTZ`, based on Gianni/Trager/Zacharias (written by Gerhard Pfister) and `primdecSY`, based on Shimoyama/Yokoyama (written by Wolfram Decker and Hans Schoenemann).

The result of `primdecGTZ` and `primdecSY` is returned as a list of pairs of ideals, where the second ideal is the prime ideal and the first ideal the corresponding primary ideal.

```
    LIB "primdec.lib";
    ring r = 0,(a,b,c,d,e,f),dp;
    ideal i= f3, ef2, e2f, bcf-adf, de+cf, be+af, e3;
    primdecGTZ(i);
  ↦ [1]:
  ↦    [1]:
  ↦       _[1]=f
  ↦       _[2]=e
  ↦    [2]:
  ↦       _[1]=f
  ↦       _[2]=e
  ↦ [2]:
  ↦    [1]:
  ↦       _[1]=f3
  ↦       _[2]=ef2
  ↦       _[3]=e2f
  ↦       _[4]=e3
  ↦       _[5]=de+cf
  ↦       _[6]=be+af
  ↦       _[7]=-bc+ad
  ↦    [2]:
  ↦       _[1]=f
  ↦       _[2]=e
  ↦       _[3]=-bc+ad
    // We consider now the ideal J of the base space of the
    // miniversal deformation of the cone over the rational
    // normal curve computed in section *8* and compute
    // its primary decomposition.
    ring R = 0,(A,B,C,D),dp;
    ideal J = CD, BD+D2, AD;
    primdecGTZ(J);
  ↦ [1]:
  ↦    [1]:
  ↦       _[1]=D
  ↦    [2]:
  ↦       _[1]=D
  ↦ [2]:
  ↦    [1]:
  ↦       _[1]=C
  ↦       _[2]=B+D
  ↦       _[3]=A
  ↦    [2]:
  ↦       _[1]=C
  ↦       _[2]=B+D
  ↦       _[3]=A
    // We see that there are two components which are both
    // prime, even linear subspaces, one 3-dimensional,
    // the other 1-dimensional.
    // (This is Pinkhams example and was the first known
    // surface singularity with two components of
    // different dimensions)
    //
    // Let us now produce an embedded component in the last
    // example, compute the minimal associated primes and
    // the radical. We use the Characteristic set methods
    // from primdec.lib.
    J = intersect(J,maxideal(3));
    // The following shows that the maximal ideal defines an embedded
    // (prime) component.
    primdecSY(J);
  ↦ [1]:
  ↦    [1]:
  ↦       _[1]=D
  ↦    [2]:
  ↦       _[1]=D
  ↦ [2]:
  ↦    [1]:
  ↦       _[1]=C
  ↦       _[2]=B+D
  ↦       _[3]=A
  ↦    [2]:
  ↦       _[1]=C
  ↦       _[2]=B+D
  ↦       _[3]=A
  ↦ [3]:
  ↦    [1]:
  ↦       _[1]=D2
  ↦       _[2]=C2
  ↦       _[3]=B2
  ↦       _[4]=AB
  ↦       _[5]=A2
  ↦       _[6]=BCD
  ↦       _[7]=ACD
  ↦    [2]:
  ↦       _[1]=D
  ↦       _[2]=C
  ↦       _[3]=B
  ↦       _[4]=A
    minAssChar(J);
  ↦ [1]:
  ↦    _[1]=C
  ↦    _[2]=B+D
  ↦    _[3]=A
  ↦ [2]:
  ↦    _[1]=D
    radical(J);
  ↦ _[1]=CD
  ↦ _[2]=BD+D2
  ↦ _[3]=AD
```

A.3.10 Normalization

The normalization will be computed for a reduced ring R/I . The result is a list of rings; ideals are always called `norid` in the rings of this list. The normalization of R/I is the product of the factor rings of the rings in the list divided out by the ideals `norid`.

```
  LIB "normal.lib";
  // ----- first example: rational quadruple point -----
  ring R=32003,(x,y,z),wp(3,5,15);
  ideal I=z*(y3-x5)+x10;
  list pr=normal(I);
↦
↦ // 'normal' created a list, say nor, of two elements.
↦ // To see the list type
↦       nor;
↦
↦ // * nor[1] is a list of 1 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦
↦ // * nor[2] is a list of 1 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
  def S=pr[1][1];
  setring S;
  norid;
↦ norid[1]=T(2)*x+y*z
↦ norid[2]=T(1)*x^2-T(2)*y
↦ norid[3]=-T(1)*y+x^7-x^2*z
↦ norid[4]=T(1)*y^2*z+T(2)*x^8-T(2)*x^3*z
↦ norid[5]=T(1)^2+T(2)*z+x^4*y*z
↦ norid[6]=T(1)*T(2)+x^6*z-x*z^2
↦ norid[7]=T(2)^2+T(1)*x*z
↦ norid[8]=x^10-x^5*z+y^3*z
  // ----- second example: union of straight lines -----
  ring R1=0,(x,y,z),dp;
  ideal I=(x-y)*(x-z)*(y-z);
  list qr=normal(I);
↦
↦ // 'normal' created a list, say nor, of two elements.
↦ // To see the list type
↦       nor;
↦
↦ // * nor[1] is a list of 2 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦
↦ // * nor[2] is a list of 2 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
  def S1=qr[1][1]; def S2=qr[1][2];
  setring S1; norid;
↦ norid[1]=-T(1)*y+T(1)*z+x-z
↦ norid[2]=T(1)*x-T(1)*y
↦ norid[3]=T(1)^2-T(1)
↦ norid[4]=x^2-x*y-x*z+y*z
  setring S2; norid;
↦ norid[1]=y-z
```

A.3.11 Kernel of module homomorphisms

Let A , B be two matrices of size $m \times r$ and $m \times s$ over the ring R and consider the corresponding maps

$$R^r \xrightarrow{A} R^m \xleftarrow{B} R^s \ .$$

We want to compute the kernel of the map $R^r \xrightarrow{A} R^m \longrightarrow R^m/\mathrm{Im}(B)$. This can be done using the modulo command:

$$\texttt{modulo}(A,B) = \ker(R^r \xrightarrow{A} R^m/\mathrm{Im}(B)) \ .$$

More precisely, the output of modulo(A,B) is a module such that the given generating vectors span the kernel on the right-hand side.

```
  ring r=0,(x,y,z),(c,dp);
  matrix A[2][2]=x,y,z,1;
  matrix B[2][2]=x2,y2,z2,xz;
  print(B);
↦ x2,y2,
↦ z2,xz
  def C=modulo(A,B);
  print(C);            // matrix of generators for the kernel
↦ yz2-x2, xyz-y2,  x2z-xy, x3-y2z,
↦ x2z-xz2,-x2z+y2z,xyz-yz2,0
  print(A*matrix(C));  // should be in Im(B)
↦ x2yz-x3,y3z-xy2, x3z+xy2z-y2z2-x2y,x4-xy2z,
↦ yz3-xz2,xyz2-x2z,x2z2-yz2,         x3z-y2z2
```

A.3.12 Algebraic dependence

Let g, f_1, …, $f_r \in K[x_1,\ldots,x_n]$. We want to check whether

1. f_1, …, f_r are algebraically dependent.

   Let $I = \langle Y_1 - f_1, \ldots, Y_r - f_r \rangle \subseteq K[x_1,\ldots,x_n,Y_1,\ldots,Y_r]$. Then $I \cap K[Y_1,\ldots,Y_r]$ are the algebraic relations between f_1, …, f_r.

2. $g \in K[f_1, \ldots, f_r]$.
   $g \in K[f_1, \ldots, f_r]$ if and only if the normal form of g with respect to I and a block ordering with respect to $X = (x_1, \ldots, x_n)$ and $Y = (Y_1, \ldots, Y_r)$ with $X > Y$ is in $K[Y]$.

Both questions can be answered using the following procedure. If the second argument is zero, it checks for algebraic dependence and returns the ideal of relations between the generators of the given ideal. Otherwise it checks for subring membership and returns the normal form of the second argument with respect to the ideal I.

```
proc algebraicDep(ideal J, poly g)
{
  def R=basering;         // give a name to the basering
  int n=size(J);
  int k=nvars(R);
  int i;
  intvec v;

  // construction of the new ring:

  // construct a weight vector
  v[n+k]=0;         // gives a zero vector of length n+k
  for(i=1;i<=k;i++)
  {
    v[i]=1;
  }
  string orde="(a("+string(v)+"),dp);";
  string ri="ring Rhelp=("+charstr(R)+"),
                        ("+varstr(R)+",Y(1.."+string(n)+")),"+orde;
                          // ring definition as a string
  execute(ri);            // execution of the string

  // construction of the new ideal I=(J[1]-Y(1),...,J[n]-Y(n))
  ideal I=imap(R,J);
  for(i=1;i<=n;i++)
  {
    I[i]=I[i]-var(k+i);
  }
  poly g=imap(R,g);
  if(g==0)
  {
    // construction of the ideal of relations by elimination
    poly el=var(1);
    for(i=2;i<=k;i++)
    {
      el=el*var(i);
    }
    ideal KK=eliminate(I,el);
    keepring(Rhelp);
    return(KK);
  }
  // reduction of g with respect to I
  ideal KK=reduce(g,std(I));
  keepring(Rhelp);
  return(KK);
  }

  // applications of the procedure
  ring r=0,(x,y,z),dp;
  ideal i=xz,yz;
  algebraicDep(i,0);
↦ _[1]=0
  // Note: after call of algebraicDep(), the basering is Rhelp.
  setring r; kill Rhelp;
  ideal j=xy+z2,z2+y2,x2y2-2xy3+y4;
  algebraicDep(j,0);
↦ _[1]=Y(1)^2-2*Y(1)*Y(2)+Y(2)^2-Y(3)
  setring r; kill Rhelp;
  poly g=y2z2-xz;
  algebraicDep(i,g);
↦ _[1]=Y(2)^2-Y(1)
  // this shows that g is contained in i.
  setring r; kill Rhelp;
  algebraicDep(j,g);
↦ _[1]=-z^4+z^2*Y(2)-x*z
  // this shows that g is contained in j.
```

A.4 Singularity Theory

A.4.1 Milnor and Tjurina number

The Milnor number, resp. the Tjurina number, of a power series f in $K[[x_1,\ldots,x_n]]$ is

$$\mathrm{milnor}(f)=\dim_K(K[[x_1,\ldots,x_n]]/\mathrm{jacob}(f)),$$

respectively

$$\mathrm{tjurina}(f)=\dim_K(K[[x_1,\ldots,x_n]]/((f)+\mathrm{jacob}(f)))$$

where `jacob(f)` is the ideal generated by the partials of `f`. `tjurina(f)` is finite, if and only if `f` has an isolated singularity. The same holds for `milnor(f)` if K has characteristic 0. Singular displays -1 if the dimension is infinite.

Singular cannot compute with infinite power series. But it can work in $\mathrm{Loc}_{(x)}K[x_1,\ldots,x_n]$, the localization of $K[x_1,\ldots,x_n]$ at the maximal ideal $(x_1,\ldots,x_n)$. To do this, one has to define a ring with a local monomial ordering such as ds, Ds, ls, ws, Ws (the second letter 's' referring to power 's'eries), or an appropriate matrix ordering. See Section B.2 [Monomial orderings], page 768 for a menu of possible orderings.

For theoretical reasons, the vector space dimension computed over the localization ring coincides with the Milnor (resp. Tjurina) number as defined above (in the power series ring).

We show in the example below the following:

- set option `prot` to have a short protocol during standard basis computation
- define the ring `r1` of characteristic 32003 with variables `x,y,z`, monomial ordering `ds`, series ring (i.e., K[x,y,z] localized at (x,y,z))
- list the information about `r1` by typing its name
- define the integers `a,b,c,t`
- define a polynomial `f` (depending on `a,b,c,t`) and display it

- define the jacobian ideal `i` of `f`
- compute a standard basis of `i`
- compute the Milnor number (=250) with `vdim` and create and display a string in order to comment the result (text between quotes `" ";` is a 'string')
- compute a standard basis of `i+(f)`
- compute the Tjurina number (=195) with `vdim`
- then compute the Milnor number (=248) and the Tjurina number (=195) for `t`=1
- reset the option to `noprot`

See also Section D.6.20 [sing_lib], page 1755 for the library commands for the computation of the Milnor and Tjurina number.

```
  option(prot);
  ring r1 = 32003,(x,y,z),ds;
  r1;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering ds
↦ //                  : names    x y z
↦ //        block   2 : ordering C
  int a,b,c,t=11,5,3,0;
  poly f = x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^(c-1)*y^(c-1)*z3+
           x^(c-2)*y^c*(y^2+t*x)^2;
  f;
↦ y5+x5y2+x2y2z3+xy7+z9+x11
  ideal i=jacob(f);
  i;
↦ i[1]=5x4y2+2xy2z3+y7+11x10
↦ i[2]=5y4+2x5y+2x2yz3+7xy6
↦ i[3]=3x2y2z2+9z8
  ideal j=std(i);
↦ 7(2)s8s10s11s12s(3)s13(4)s(5)s14(6)s(7)15--.s(6)-16.-.s(5)17.s(7)s--s18(6\
   ).--19-..sH(24)20(3)...21....22....23.--24-
↦ product criterion:10 chain criterion:69
  "The Milnor number of f(11,5,3) for t=0 is", vdim(j);
↦ The Milnor number of f(11,5,3) for t=0 is 250
  j=i+f;    // override j
  j=std(j);
↦ 7(3)s8(2)s10s11(3)ss12(4)s(5)s13(6)s(8)s14(9).s(10).15--sH(23)(8)...16...\
   ...17.......sH(21)18(9)sH(20)(8)s17(10)..18..-.......19..---..sH(19)
↦ product criterion:11 chain criterion:62
  vdim(j);  // compute the Tjurina number for t=0
↦ 195
  t=1;
  f=x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^(c-1)*y^(c-1)*z3
    +x^(c-2)*y^c*(y^2+t*x)^2;
  ideal i1=jacob(f);
  ideal j1=std(i1);
↦ 7(2)s8s10s11s12s13(3)ss(4)s14(5)s(6)s15(7).....s(8)16.s...s(9)..17.......\
   .....s18(10).....s(11)..-.19.......sH(24)(10).....20...........21........\
   ..22.............................23...............................24.----\
   ------.25.26
↦ product criterion:11 chain criterion:83
```

```
  "The Milnor number of f(11,5,3) for t=1:",vdim(j1);
↦ The Milnor number of f(11,5,3) for t=1: 248
  vdim(std(j1+f));   // compute the Tjurina number for t=1
↦ 7(16)s8(15)s10s11ss(16)-12.s-s13s(17)s(18)s(19)-s(18).-14-s(17)-s(16)ss(1\
   7)s15(18)..-s...--.16....-.......s(16).sH(23)s(18)...17.........18.......\
   ...sH(20)17(17)18...19..--...-.-...-..........20.-----s17(9)..18...19..-2\
   0.-.............21..........sH(19)(4)----
↦ product criterion:15 chain criterion:174
↦ 195
  option(noprot);
```

A.4.2 Critical points

The same computation which computes the Milnor, resp. the Tjurina, number, but with ordering `dp` instead of `ds` (i.e., in $K[x_1,\ldots,x_n]$ instead of $\mathrm{Loc}_{(x)}K[x_1,\ldots,x_n]$) gives:

- the number of critical points of `f` in the affine space (counted with multiplicities)
- the number of singular points of `f` on the affine hypersurface `f`=0 (counted with multiplicities).

We start with the ring `r1` from section Section A.4.1 [Milnor and Tjurina number], page 734 and its elements.

The following will be implemented below:

- reset the protocol option and activate the timer
- define the ring `r2` of characteristic 32003 with variables `x,y,z` and monomial ordering `dp` (= degrevlex) (i.e., the polynomial ring = K[x,y,z]).
- Note that polynomials, ideals, matrices (of polys), vectors, modules belong to a ring, hence we have to define `f` and `jacob(f)` again in `r2`. Since these objects are local to a ring, we may use the same names. Instead of defining `f` again we map it from ring `r1` to `r2` by using the `imap` command (`imap` is a convenient way to map variables from some ring identically to variables with the same name in the basering, even if the ground field is different. Compare with `fetch` which works for almost identical rings, e.g., if the rings differ only by the ordering or by the names of the variables and which may be used to rename variables). Integers and strings, however, do not belong to any ring. Once defined they are globally known.
- The result of the computation here (together with the previous one in Section A.4.1 [Milnor and Tjurina number], page 734) shows that (for `t`=0) $\dim_K(\mathrm{Loc}_{(x,y,z)}K[x,y,z]/\mathrm{jacob}(f))$ = 250 (previously computed) while $\dim_K(K[x,y,z]/\mathrm{jacob}(f)) = 536$. Hence `f` has 286 critical points, counted with multiplicity, outside the origin. Moreover, since $\dim_K(\mathrm{Loc}_{(x,y,z)}K[x,y,z]/(\mathrm{jacob}(f)+(f))) = 195 = \dim_K(K[x,y,z]/(\mathrm{jacob}(f)+(f)))$, the affine surface `f`=0 is smooth outside the origin.

```
  ring r1 = 32003,(x,y,z),ds;
  int a,b,c,t=11,5,3,0;
  poly f = x^a+y^b+z^(3*c)+x^(c+2)*y^(c-1)+x^(c-1)*y^(c-1)*z3+
           x^(c-2)*y^c*(y^2+t*x)^2;
  option(noprot);
  timer=1;
  ring r2 = 32003,(x,y,z),dp;
  poly f=imap(r1,f);
  ideal j=jacob(f);
  vdim(std(j));
↦ 536
  vdim(std(j+f));
↦ 195
```

```
  timer=0;  // reset timer
```

A.4.3 Polar curves

The polar curve of a hypersurface given by a polynomial $f \in k[x_1, \ldots, x_n, t]$ with respect to t (we may consider $f = 0$ as a family of hypersurfaces parametrized by t) is defined as the Zariski closure of $V(\partial f/\partial x_1, \ldots, \partial f/\partial x_n) \setminus V(f)$ if this happens to be a curve. Some authors consider $V(\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$ itself as polar curve.

We may consider projective hypersurfaces (in P^n), affine hypersurfaces (in k^n) or germs of hypersurfaces (in $(k^n, 0)$), getting in this way projective, affine or local polar curves.

Now let us compute this for a family of curves. We need the library `elim.lib` for saturation and `sing.lib` for the singular locus.

```
  LIB "elim.lib";
  LIB "sing.lib";
  // Affine polar curve:
  ring R = 0,(x,z,t),dp;              // global ordering dp
  poly f = z5+xz3+x2-tz6;
  dim_slocus(f);                      // dimension of singular locus
↦ 1
  ideal j = diff(f,x),diff(f,z);
  dim(std(j));                        // dim V(j)
↦ 1
  dim(std(j+ideal(f)));               // V(j,f) also 1-dimensional
↦ 1
  // j defines a curve, but to get the polar curve we must remove the
  // branches contained in f=0 (they exist since dim V(j,f) = 1). This
  // gives the polar curve set theoretically. But for the structure we
  // may take either j:f or j:f^k for k sufficiently large. The first is
  // just the ideal quotient, the second the iterated ideal quotient
  // or saturation. In our case both coincide.
  ideal q = quotient(j,ideal(f));     // ideal quotient
  ideal qsat = sat(j,f);              // saturation, proc from elim.lib
  ideal sq = std(q);
  dim(sq);
↦ 1
  // 1-dimensional, hence q defines the affine polar curve
  //
  // to check that q and qsat are the same, we show both inclusions, i.e.,
  // both reductions must give the 0-ideal
  size(reduce(qsat,sq));
↦ 0
  size(reduce(q,std(qsat)));
↦ 0
  qsat;
↦ qsat[1]=12zt+3z-10
↦ qsat[2]=5z2+12xt+3x
↦ qsat[3]=144xt2+72xt+9x+50z
  // We see that the affine polar curve does not pass through the origin,
  // hence we expect the local polar "curve" to be empty
  // ------------------------------------------------
  // Local polar curve:
  ring r = 0,(x,z,t),ds;              // local ordering ds
  poly f = z5+xz3+x2-tz6;
  ideal j = diff(f,x),diff(f,z);
  dim(std(j));                        // V(j) 1-dimensional
↦ 1
  dim(std(j+ideal(f)));               // V(j,f) also 1-dimensional
↦ 1
  ideal q = quotient(j,ideal(f));     // ideal quotient
  q;
↦ q[1]=1
  // The local polar "curve" is empty, i.e., V(j) is contained in V(f)
  // -----------------------------------------------
  // Projective polar curve: (we need "sing.lib" and "elim.lib")
  ring P = 0,(x,z,t,y),dp;            // global ordering dp
  poly f = z5y+xz3y2+x2y4-tz6;
                                      // but consider t as parameter
  dim_slocus(f);              // projective 1-dimensional singular locus
↦ 2
  ideal j = diff(f,x),diff(f,z);
  dim(std(j));                        // V(j), projective 1-dimensional
↦ 2
  dim(std(j+ideal(f)));               // V(j,f) also projective 1-dimensional
↦ 2
  ideal q = quotient(j,ideal(f));
  ideal qsat = sat(j,f);              // saturation, proc from elim.lib
  dim(std(qsat));
↦ 2
  // projective 1-dimensional, hence q and/or qsat define the projective
  // polar curve. In this case, q and qsat are not the same, we needed
  // 2 quotients.
  // Let us check both reductions:
  size(reduce(qsat,std(q)));
↦ 4
  size(reduce(q,std(qsat)));
↦ 0
  // Hence q is contained in qsat but not conversely
  q;
↦ q[1]=12zty+3zy-10y2
↦ q[2]=60z2t-36xty-9xy-50zy
↦ q[3]=12xty2+5z2y+3xy2
↦ q[4]=z3y+2xy3
  qsat;
↦ qsat[1]=12zt+3z-10y
↦ qsat[2]=12xty+5z2+3xy
↦ qsat[3]=144xt2+72xt+9x+50z
↦ qsat[4]=z3+2xy2
  //
  // Now consider again the affine polar curve,
  // homogenize it with respect to y (deg t=0) and compare:
  // affine polar curve:
  ideal qa = 12zt+3z-10,5z2+12xt+3x,-144xt2-72xt-9x-50z;
  // homogenized:
  ideal qh = 12zt+3z-10y,5z2+12xyt+3xy,-144xt2-72xt-9x-50z;
  size(reduce(qh,std(qsat)));
```

```
↦ 0
  size(reduce(qsat,std(qh)));
↦ 0
  // both ideals coincide
```

A.4.4 T1 and T2

T^1 , resp. T^2 , of an ideal j usually denote the modules of infinitesimal deformations, resp. of obstructions. In Singular there are procedures `T_1` and `T_2` in `sing.lib` such that `T_1(j)` and `T_2(j)` compute a standard basis of a presentation of these modules. If T^1, T_2 are finite dimensional K-vector spaces (e.g., for isolated singularities), a basis can be computed by applying `kbase(T_1(j));`, resp. `kbase(T_2(j));`, the dimensions by applying `vdim`. For a complete intersection j the procedure `Tjurina` also computes T^1 , but faster ($T^2 = 0$ in this case). For a non complete intersection, it is faster to use the procedure `T_12` instead of `T_1` and `T_2`. Type `help T_1;` (or `help T_2;` or `help T_12;`) to obtain more detailed information about these procedures.

We give three examples, the first being a hypersurface, the second a complete intersection, the third not a complete intersection:

- load `sing.lib`
- check whether the ideal j is a complete intersection. It is, if number of variables = dimension + minimal number of generators
- compute the Tjurina number
- compute a vector space basis (kbase) of T^1
- compute the Hilbert function of T^1
- create a polynomial encoding the Hilbert series
- compute the dimension of T^2

```
  LIB "sing.lib";
  ring R=32003,(x,y,z),ds;
  // ---------------------------------------
  // hypersurface case (from series T[p,q,r]):
  int p,q,r = 3,3,4;
  poly f = x^p+y^q+z^r+xyz;
  tjurina(f);
↦ 8
  // Tjurina number = 8
  kbase(Tjurina(f));
↦ // Tjurina number = 8
↦ _[1]=z3
↦ _[2]=z2
↦ _[3]=yz
↦ _[4]=xz
↦ _[5]=z
↦ _[6]=y
↦ _[7]=x
↦ _[8]=1
  // ---------------------------------------
  // complete intersection case (from series P[k,l]):
  int k,l =3,2;
  ideal j=xy,x^k+y^l+z2;
  dim(std(j));          // Krull dimension
↦ 1
```

```
  size(minbase(j));     // minimal number of generators
↦ 2
  tjurina(j);           // Tjurina number
↦ 6
  module T=Tjurina(j);
↦ // Tjurina number = 6
  kbase(T);             // a sparse output of the k-basis of T_1
↦ _[1]=z*gen(1)
↦ _[2]=gen(1)
↦ _[3]=y*gen(2)
↦ _[4]=x2*gen(2)
↦ _[5]=x*gen(2)
↦ _[6]=gen(2)
  print(kbase(T));      // columns of matrix are a k-basis of T_1
↦ z,1,0,0, 0,0,
↦ 0,0,y,x2,x,1
  // --------------------------------------
  // general case (cone over rational normal curve of degree 4):
  ring r1=0,(x,y,z,u,v),ds;
  matrix m[2][4]=x,y,z,u,y,z,u,v;
  ideal i=minor(m,2);   // 2x2 minors of matrix m
  module M=T_1(i);       // a presentation matrix of T_1
↦ // dim T_1 = 4
  vdim(M);              // Tjurina number
↦ 4
  hilb(M);              // display of both Hilbert series
↦ (-4t5+20t4-40t3+40t2-20t+4) / (1-t)^5
↦ (4) / (1-t)^0
↦ // dimension (local)   = 0
↦ // multiplicity = 4
  intvec v1=hilb(M,1);  // first Hilbert series as intvec
  intvec v2=hilb(M,2);  // second Hilbert series as intvec
  v1;
↦ 4,-20,40,-40,20,-4,0
  v2;
↦ 4,0
  v1[3];                // 3rd coefficient of the 1st Hilbert series
↦ 40
  module N=T_2(i);
↦ // dim T_2 = 3
// In some cases it might be useful to have a polynomial in some ring
// encoding the Hilbert series. This polynomial can then be
// differentiated, evaluated etc. It can be done as follows:
ring H = 0,t,ls;
poly h1;
int ii;
for (ii=1; ii<=size(v1); ii=ii+1)
{
  h1=h1+v1[ii]*t^(ii-1);
}
h1;                     // 1st Hilbert series
↦ 4-20t+40t2-40t3+20t4-4t5
diff(h1,t);             // differentiate  h1
```

```
↦ -20+80t-120t2+80t3-20t4
subst(h1,t,1);        // substitute t by 1
↦ 0

// The procedures T_1, T_2, T_12 may be called with two arguments and then
// they return a list with more information (type help T_1; etc.)
// e.g., T_12(i,<any>); returns a list with 9 nonempty objects where
// _[1] = std basis of T_1-module, _[2] = std basis of T_2-module,
// _[3]= vdim of T_1, _[4]= vdim of T_2
setring r1;           // make r1 again the basering
list L = T_12(i,1);
↦ // dim T_1  =  4
↦ // dim T_2  =  3
kbase(L[1]);          // kbase of T_1
↦ _[1]=1*gen(2)
↦ _[2]=1*gen(3)
↦ _[3]=1*gen(6)
↦ _[4]=1*gen(7)
kbase(L[2]);          // kbase of T_2
↦ _[1]=1*gen(6)
↦ _[2]=1*gen(8)
↦ _[3]=1*gen(9)
L[3];                 // vdim of T_1
↦ 4
L[4];                 // vdim of T_2
↦ 3
```

A.4.5 Deformations

- The libraries sing.lib, respextively deform.lib, contain procedures to compute total and base space of the miniversal (= semiuniversal) deformation of an isolated complete intersection singularity, respectively of an arbitrary isolated singularity.
- The procedure deform in sing.lib returns a matrix whose columns $h_1, \ldots, h_r$ represent all 1st order deformations. More precisely, if $I \subset R$ is the ideal generated by $f_1, ..., f_s$, then any infinitesimal deformation of R/I over $K[\varepsilon]/(\varepsilon^2)$ is given by $f + \varepsilon g$, where $f = (f_1, ..., f_s)$, and where g is a K-linear combination of the h_i.
- The procedure versal in deform.lib computes a formal miniversal deformation up to a certain order which can be prescribed by the user. For a complete intersection the 1st order part is already miniversal.
- The procedure versal extends the basering to a new ring with additional deformation parameters which contains the equations for the miniversal base space and the miniversal total space.
- There are default names for the objects created, but the user may also choose their own names.
- If the user sets printlevel=2; before running versal, some intermediate results are shown. This is useful since versal is already complicated and might run for some time on more complicated examples. (type help versal;)

We compute for the same examples as in the section Section A.4.4 [T1 and T2], page 739 the miniversal deformations:

```
LIB "deform.lib";
ring R=32003,(x,y,z),ds;
```

```
  //---------------------------------------------------
  // hypersurface case (from series T[p,q,r]):
  int p,q,r = 3,3,4;
  poly f = x^p+y^q+z^r+xyz;
  print(deform(f));
↦ z3,z2,yz,xz,z,y,x,1
  // the miniversal deformation of f=0 is the projection from the
  // miniversal total space to the miniversal base space:
  // { (A,B,C,D,E,F,G,H,x,y,z) | x3+y3+xyz+z4+A+Bx+Cxz+Dy+Eyz+Fz+Gz2+Hz3 =0 }
  //  --> { (A,B,C,D,E,F,G,H) }
  //---------------------------------------------------
  // complete intersection case (from series P[k,l]):
  int k,l =3,2;
  ideal j=xy,x^k+y^l+z2;
  print(deform(j));
↦ 0,0, 0,0,z,1,
↦ y,x2,x,1,0,0
  def L=versal(j);            // using default names
↦ // smooth base space
↦ // ready: T_1 and T_2
↦ 
↦ 
↦ // 'versal' returned a list, say L, of four rings. In L[1] are stored:
↦ //   as matrix Fs: Equations of total space of the miniversal deformation\
   ,
↦ //   as matrix Js: Equations of miniversal base space,
↦ //   as matrix Rs: syzygies of Fs mod Js.
↦ // To access these data, type
↦      def Px=L[1]; setring Px; print(Fs); print(Js); print(Rs);
↦ 
↦ // L[2] = L[1]/Fo extending Qo=Po/Fo,
↦ // L[3] = the embedding ring of the versal base space,
↦ // L[4] = L[1]/Js extending L[3]/Js.
↦ 
  def Px=L[1]; setring Px;
  show(Px);                   // show is a procedure from inout.lib
↦ // ring: (ZZ/32003),(A,B,C,D,E,F,x,y,z),(ds(6),ds(3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦ // Rs                             [0]  matrix 2 x 1
↦ // Fs                             [0]  matrix 1 x 2
↦ // Js                             [0]  matrix 1 x 0
  listvar(matrix);
↦ // Rs                             [0]  matrix 2 x 1
↦ // Fs                             [0]  matrix 1 x 2
↦ // Js                             [0]  matrix 1 x 0
  // ___ Equations of miniversal base space ___:
  Js;
↦ 
  // ___ Equations of miniversal total space ___:
  Fs;
↦ Fs[1,1]=y2+z2+x3+Cy+Dx2+Ex+F
↦ Fs[1,2]=xy+Az+B
```

```
  // the miniversal deformation of V(j) is the projection from the
  // miniversal total space to the miniversal base space:
  // { (A,B,C,D,E,F,x,y,z) | xy+F+Ez=0, y2+z2+x3+D+Cx+Bx2+Ay=0 }
  //  --> { (A,B,C,D,E,F) }
  //----------------------------------------------------
  // general case (cone over rational normal curve of degree 4):
  kill L;
  ring r1=0,(x,y,z,u,v),ds;
  matrix m[2][4]=x,y,z,u,y,z,u,v;
  ideal i=minor(m,2);                 // 2x2 minors of matrix m
  int time=timer;
  // Call parameters of the miniversal base A(1),A(2),...:
  def L=versal(i,0,"","A(");
↦ // ready: T_1 and T_2
↦ // start computation in degree 2.
↦
↦
↦ // 'versal' returned a list, say L, of four rings. In L[1] are stored:
↦ //   as matrix Fs: Equations of total space of the miniversal deformation\
  ,
↦ //   as matrix Js: Equations of miniversal base space,
↦ //   as matrix Rs: syzygies of Fs mod Js.
↦ // To access these data, type
↦      def Px=L[1]; setring Px; print(Fs); print(Js); print(Rs);
↦
↦ // L[2] = L[1]/Fo extending Qo=Po/Fo,
↦ // L[3] = the embedding ring of the versal base space,
↦ // L[4] = L[1]/Js extending L[3]/Js.
↦
  "// used time:",timer-time,"sec";   // time of last command
↦ // used time: 0 sec
  def Def_rPx=L[1]; setring Def_rPx;
  Fs;
↦ Fs[1,1]=u^2-z*v-A(2)*u+A(4)*v
↦ Fs[1,2]=z*u-y*v-A(1)*u+A(4)*u
↦ Fs[1,3]=y*u-x*v+A(3)*u+A(4)*z
↦ Fs[1,4]=z^2-y*u-A(1)*z+A(2)*y
↦ Fs[1,5]=y*z-x*u+A(2)*x+A(3)*z
↦ Fs[1,6]=y^2-x*z+A(1)*x+A(3)*y
  Js;
↦ Js[1,1]=A(2)*A(4)
↦ Js[1,2]=-A(1)*A(4)+A(4)^2
↦ Js[1,3]=A(3)*A(4)
  // the miniversal deformation of V(i) is the projection from the
  // miniversal total space to the miniversal base space:
  // { (A(1..4),x,y,z,u,v) |
  //         -u^2+x*v+A(2)*u+A(4)*v=0, -z*u+y*v-A(1)*u+A(3)*u=0,
  //         -y*u+x*v+A(3)*u+A(4)*z=0,  z^2-y*u+A(1)*z+A(2)*y=0,
  //          y*z-x*u+A(2)*x-A(3)*z=0, -y^2+x*z+A(1)*x+A(3)*y=0 }
  //  --> { A(1..4) |
  //          A(2)*A(4) = -A(3)*A(4) = -A(1)*A(4)+A(4)^2 = 0 }
  //----------------------------------------------------
```

A.4.6 Invariants of plane curve singularities

The Puiseux pairs of an irreducible and reduced plane curve singularity are probably its most important invariants. They can be computed from its Hamburger-Noether expansion (which is the analogue of the Puiseux expansion in characteristic 0 for fields of arbitrary characteristic).

The library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718) uses the algorithm of Antonio Campillo in "Algebroid curves in positive characteristic" SLN 813, 1980. This algorithm has the advantage that it needs least possible field extensions and, moreover, works in any characteristic. This fact can be used to compute the invariants over a field of finite characteristic, say 32003, which will most probably be the same as in characteristic 0.

We compute the Hamburger-Noether expansion of a plane curve singularity given by a polynomial `f` in two variables. This expansion is given by a matrix, and it allows us to compute a primitive parametrization (up to a given order) for the curve singularity defined by `f` and numerical invariants such as the

- characteristic exponents,
- Puiseux pairs (of a complex model),
- degree of the conductor,
- delta invariant,
- generators of the semigroup.

Besides commands for computing a parametrization and the invariants mentioned above, the library `hnoether.lib` provides commands for the computation of the Newton polygon of `f`, the square-free part of `f` and a procedure to convert one set of invariants to another.

```
  LIB "hnoether.lib";
  // ======== The irreducible case ========
  ring s = 0,(x,y),ds;
  poly f = y4-2x3y2-4x5y+x6-x7;
  list hn = develop(f);
  show(hn[1]);     // Hamburger-Noether matrix
↦ // matrix, 3x3
↦ 0,x,  0,
↦ 0,1,  x,
↦ 0,1/4,-1/2
  displayHNE(hn);  // Hamburger-Noether development
↦   y = z(1)*x
↦   x = z(1)^2+z(1)^2*z(2)
↦   z(1) = 1/4*z(2)^2-1/2*z(2)^3 + ..... (terms of degree >=4)
  setring s;
  displayInvariants(hn);
↦  characteristic exponents  : 4,6,7
↦  generators of semigroup   : 4,6,13
↦  Puiseux pairs            : (3,2)(7,2)
↦  degree of the conductor  : 16
↦  delta invariant          : 8
↦  sequence of multiplicities: 4,2,2,1,1
  // invariants(hn);  returns the invariants as list
  // partial parametrization of f: param takes the first variable
  // as infinite except the ring has more than 2 variables. Then
  // the 3rd variable is chosen.
  param(hn);
↦ // ** Warning: result is exact up to order 5 in x and 7 in y !
↦ _[1]=1/16x4-3/16x5+1/4x7
↦ _[2]=1/64x6-5/64x7+3/32x8+1/16x9-1/8x10
  ring extring=0,(x,y,t),ds;
  poly f=x3+2xy2+y2;
  list hn=develop(f,-1);
  param(hn);       // partial parametrization of f
↦ // ** Warning: result is exact up to order 2 in x and 3 in y !
↦ _[1]=-t2
↦ _[2]=-t3
  list hn1=develop(f,6);
  param(hn1);     // a better parametrization
↦ // ** Warning: result is exact up to order 6 in x and 7 in y !
↦ _[1]=-t2+2t4-4t6
↦ _[2]=-t3+2t5-4t7
  // instead of recomputing you may extend the development:
  list hn2=extdevelop(hn,12);
  param(hn2);     // a still better parametrization
↦ // ** Warning: result is exact up to order 12 in x and 13 in y !
↦ _[1]=-t2+2t4-4t6+8t8-16t10+32t12
↦ _[2]=-t3+2t5-4t7+8t9-16t11+32t13
  //
  // ======== The reducible case ========
  ring r = 0,(x,y),dp;
  poly f=x11-2y2x8-y3x7-y2x6+y4x5+2y4x3+y5x2-y6;
  // = (x5-1y2) * (x6-2x3y2-1x2y3+y4)
  list L=hnexpansion(f);
↦ // No change of ring necessary, return value is HN expansion.
  show(L[1][1]);     // Hamburger-Noether matrix of 1st branch
↦ // matrix, 3x3
↦ 0,x,0,
↦ 0,1,x,
↦ 0,1,-1
  displayInvariants(L);
↦  --- invariants of branch number 1 : ---
↦  characteristic exponents  : 4,6,7
↦  generators of semigroup   : 4,6,13
↦  Puiseux pairs             : (3,2)(7,2)
↦  degree of the conductor   : 16
↦  delta invariant           : 8
↦  sequence of multiplicities: 4,2,2,1,1
↦
↦  --- invariants of branch number 2 : ---
↦  characteristic exponents  : 2,5
↦  generators of semigroup   : 2,5
↦  Puiseux pairs             : (5,2)
↦  degree of the conductor   : 4
↦  delta invariant           : 2
↦  sequence of multiplicities: 2,2,1,1
↦
↦  -------------- contact numbers : --------------
↦
↦ branch |    2
↦ -------+-----
↦     1  |    2
↦ 
↦  ------------- intersection multiplicities : -------------
↦ 
↦ branch |    2
↦ -------+-----
↦     1  |   12
↦ 
↦  ------------- delta invariant of the curve :  22
  param(L[2]);       // parametrization of 2nd branch
↦ _[1]=x2
↦ _[2]=x5
```

A.4.7 Branches of space curve singularities

In this example, the number of branches of a given quasihomogeneous isolated space curve singularity will be computed as an example of the pitfalls appearing in the use of primary decomposition. When dealing with singularities, two situations are possible in which the primary decomposition algorithm might not lead to a complete decomposition: first of all, one of the computed components could be globally irreducible, but analytically reducible (this is impossible for quasihomogeneous singularities) and, as a second possibility, a component might be irreducible over the rational numbers, but reducible over the complex numbers.

```
  ring r=0,(x,y,z),ds;
  ideal i=x^4-y*z^2,x*y-z^3,y^2-x^3*z;  // the space curve singularity
  qhweight(i);
↦ 1,2,1
  // The given space curve singularity is quasihomogeneous. Hence we can pass
  // to the polynomial ring.
  ring rr=0,(x,y,z),dp;
  ideal i=imap(r,i);
  resolution ires=mres(i,0);
  ires;
↦   1       3       2
↦ rr <--  rr <--  rr
↦ 
↦ 0       1       2
↦ 
  // From the structure of the resolution, we see that the Cohen-Macaulay
  // type of the given singularity is 2
  //
  // Let us now look for the branches using the primdec library.
  LIB "primdec.lib";
  primdecSY(i);
↦ [1]:
↦    [1]:
↦       _[1]=z3-xy
↦       _[2]=x3+x2z+xz2+xy+yz
↦       _[3]=x2z2+x2y+xyz+yz2+y2
↦    [2]:
↦       _[1]=z3-xy
↦       _[2]=x3+x2z+xz2+xy+yz
↦       _[3]=x2z2+x2y+xyz+yz2+y2
↦ [2]:
```

```
↦    [1]:
↦       _[1]=x-z
↦       _[2]=z2-y
↦    [2]:
↦       _[1]=x-z
↦       _[2]=z2-y
  def li=_[1];
  ideal i2=li[2];       // call the first ideal i1
  // The curve seems to have 2 branches by what we computed using the
  // algorithm of Shimoyama-Yokoyama.
  // Now the same computation by the Gianni-Trager-Zacharias algorithm:
  primdecGTZ(i);
↦ [1]:
↦    [1]:
↦       _[1]=-z2+y
↦       _[2]=x-z
↦    [2]:
↦       _[1]=-z2+y
↦       _[2]=x-z
↦ [2]:
↦    [1]:
↦       _[1]=z8+yz6+y2z4+y3z2+y4
↦       _[2]=xz5+z6+yz4+y2z2+y3
↦       _[3]=-z3+xy
↦       _[4]=x2z2+xz3+xyz+yz2+y2
↦       _[5]=x3+x2z+xz2+xy+yz
↦    [2]:
↦       _[1]=z8+yz6+y2z4+y3z2+y4
↦       _[2]=xz5+z6+yz4+y2z2+y3
↦       _[3]=-z3+xy
↦       _[4]=x2z2+xz3+xyz+yz2+y2
↦       _[5]=x3+x2z+xz2+xy+yz
  // Having computed the primary decomposition in 2 different ways and
  // having obtained the same number of branches, we might expect that the
  // number of branches is really 2, but we can check this by formulae
  // for the invariants of space curve singularities:
  //
  // mu = tau - t + 1 (for quasihomogeneous curve singularities)
  // where mu denotes the Milnor number, tau the Tjurina number and
  // t the Cohen-Macaulay type
  //
  // mu = 2 delta - r + 1
  // where delta denotes the delta-Invariant and r the number of branches
  //
  // tau can be computed by using the corresponding procedure T1 from
  // sing.lib.
  setring r;
  LIB "sing.lib";
  T_1(i);
↦ // dim T_1 = 13
↦ _[1]=gen(6)+2z*gen(5)
↦ _[2]=gen(4)+3x2*gen(2)
↦ _[3]=gen(3)+gen(1)
↦ _[4]=x*gen(5)-y*gen(2)-z*gen(1)
↦ _[5]=x*gen(1)-z2*gen(2)
↦ _[6]=y*gen(5)+3x2z*gen(2)
↦ _[7]=y*gen(2)-z*gen(1)
↦ _[8]=2y*gen(1)-z2*gen(5)
↦ _[9]=z2*gen(5)
↦ _[10]=z2*gen(1)
↦ _[11]=x3*gen(2)
↦ _[12]=x2z2*gen(2)
↦ _[13]=xz3*gen(2)
↦ _[14]=z4*gen(2)
  setring rr;
  // Hence tau is 13 and therefore mu is 12. But then it is impossible that
  // the singularity has two branches, since mu is even and delta is an
  // integer!
  // So obviously, we did not decompose completely. Because the second branch
  // is smooth, only the first ideal can be the one which can be decomposed
  // further.
  // Let us now consider the normalization of this first ideal i1.
  LIB "normal.lib";
  normal(i2);
↦ 
↦ // 'normal' created a list, say nor, of two elements.
↦ // To see the list type
↦       nor;
↦ 
↦ // * nor[1] is a list of 1 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦ 
↦ // * nor[2] is a list of 1 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
↦ [1]:
↦    [1]:
↦       // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering dp
↦ //                  : names    x y z
↦ //        block   3 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=y
↦       _[2]=xz
↦        _[3]=x2
↦        _[4]=z2
  def rno=_[1][1];
  setring rno;
  norid;
↦ norid[1]=-T(2)*z+x
↦ norid[2]=T(1)*x-z
↦ norid[3]=T(2)*x-T(3)*z
↦ norid[4]=T(1)*z+T(2)*z+T(3)*x+T(3)*z+z
↦ norid[5]=-T(2)*y+z^2
↦ norid[6]=T(1)*z^2-y
↦ norid[7]=T(2)*z^2-T(3)*y
↦ norid[8]=T(1)*y+T(2)*y+T(3)*z^2+T(3)*y+y
↦ norid[9]=T(1)^2+T(1)+T(2)+T(3)+1
↦ norid[10]=T(1)*T(2)-1
↦ norid[11]=T(2)^2-T(3)
↦ norid[12]=T(1)*T(3)-T(2)
↦ norid[13]=T(2)*T(3)+T(1)+T(2)+T(3)+1
↦ norid[14]=T(3)^2-T(1)
↦ norid[15]=z^3-x*y
↦ norid[16]=x^3+x^2*z+x*z^2+x*y+y*z
↦ norid[17]=x^2*z^2+x^2*y+x*y*z+y*z^2+y^2
  // The ideal is generated by a polynomial in one variable of degree 4 which
  // factors completely into 4 polynomials of type T(2)+a.
  // From this, we know that the ring of the normalization is the direct sum of
  // 4 polynomial rings in one variable.
  // Hence our original curve has these 4 branches plus a smooth one
  // which we already determined by primary decomposition.
  // Our final result is therefore: 5 branches.
```

A.4.8 Classification of hypersurface singularities

Classification of isolated hypersurface singularities with respect to right equivalence is provided by the command classify of the library classify.lib. The classification is done by using the algorithm of Arnold. Before entering this algorithm, a first guess based on the Hilbert polynomial of the Milnor algebra is made.

```
  LIB "classify.lib";
  ring r=0,(x,y,z),ds;
  poly p=singularity("E[6k+2]",2)[1];
  p=p+z^2;
  p;
↦ z2+x3+xy6+y8
  // We received an E_14 singularity in normal form
  // from the database of normal forms. Since only the residual
  // part is saved in the database, we added z^2 to get an E_14
  // of embedding dimension 3.
  //
  // Now we apply a coordinate change in order to deal with a
  // singularity which is not in normal form:
  map phi=r,x+y,y+z,x;
  poly q=phi(p);
  // Yes, q really looks ugly, now:
  q;
```

```
↦ x2+x3+3x2y+3xy2+y3+xy6+y7+6xy5z+6y6z+15xy4z2+15y5z2+20xy3z3+20y4z3+15xy2z\
  4+15y3z4+6xyz5+6y2z5+xz6+yz6+y8+8y7z+28y6z2+56y5z3+70y4z4+56y3z5+28y2z6+8\
  yz7+z8
  // Classification
  classify(q);
↦ About the singularity :
↦           Milnor number(f)   = 14
↦           Corank(f)          = 2
↦           Determinacy       <= 12
↦ Guessing type via Milnorcode:   E[6k+2]=E[14]
↦ 
↦ Computing normal form ...
↦ I have to apply the splitting lemma. This will take some time....:-)
↦    Arnold step number 9
↦ The singularity
↦    x3-9/4x4+27/4x5-189/8x6+737/8x7+6x6y+15x5y2+20x4y3+15x3y4+6x2y5+xy6-24\
  089/64x8-x7y+11/2x6y2+26x5y3+95/2x4y4+47x3y5+53/2x2y6+8xy7+y8+104535/64x9\
  +27x8y+135/2x7y2+90x6y3+135/2x5y4+27x4y5+9/2x3y6-940383/128x10-405/4x9y-2\
  025/8x8y2-675/2x7y3-2025/8x6y4-405/4x5y5-135/8x4y6+4359015/128x11+1701/4x\
  10y+8505/8x9y2+2835/2x8y3+8505/8x7y4+1701/4x6y5+567/8x5y6-82812341/512x12\
  -15333/8x11y-76809/16x10y2-25735/4x9y3-78525/16x8y4-16893/8x7y5-8799/16x6\
  y6-198x5y7-495/4x4y8-55x3y9-33/2x2y10-3xy11-1/4y12
↦ is R-equivalent to E[14].
↦    Milnor number = 14
↦    modality      = 1
↦ 2z2+x3+xy6+y8
  // The library also provides routines to determine the corank of q
  // and its residual part without going through the whole
  // classification algorithm.
  corank(q);
↦ 2
  morsesplit(q);
↦ y3-9/4y4+27/4y5-189/8y6+737/8y7+6y6z+15y5z2+20y4z3+15y3z4+6y2z5+yz6-24089\
  /64y8-y7z+11/2y6z2+26y5z3+95/2y4z4+47y3z5+53/2y2z6+8yz7+z8+104535/64y9+27\
  y8z+135/2y7z2+90y6z3+135/2y5z4+27y4z5+9/2y3z6-940383/128y10-405/4y9z-2025\
  /8y8z2-675/2y7z3-2025/8y6z4-405/4y5z5-135/8y4z6+4359015/128y11+1701/4y10z\
  +8505/8y9z2+2835/2y8z3+8505/8y7z4+1701/4y6z5+567/8y5z6-82812341/512y12-15\
  333/8y11z-76809/16y10z2-25735/4y9z3-78525/16y8z4-16893/8y7z5-8799/16y6z6-\
  198y5z7-495/4y4z8-55y3z9-33/2y2z10-3yz11-1/4z12
```

A.4.9 Resolution of singularities

Resolution of singularities and applications thereof are provided by the libraries `resolve.lib` and `reszeta.lib`; graphical output may be generated automatically by using external programs `surf` and `dot` respectively to which a specialized interface is provided by the library `resgraph.lib`. In this example, the basic functionality of the resolution of singularities package is illustrated by the computation of the intersection matrix and genera of the exceptional curves on a surface obtained from resolving the A6 surface singularity. A separate tutorial, which introduces the complete functionality of the package and explains the rather complicated data structures appearing in intermediate results, can be found at `https://www.singular.uni-kl.de/tutor_resol.pdf`.

```
LIB"resolve.lib";                    // load the resolution algorithm
LIB"reszeta.lib";                    // load its application algorithms
```

```
ring R=0,(x,y,z),dp;              // define the ring Q[x,y,z]
ideal I=x7+y2-z2;                 // an A6 surface singularity
list L=resolve(I);                // compute the resolution
list iD=intersectionDiv(L);       // compute intersection properties
iD;                               // show the output
↦ [1]:
↦    -2,0,1,0,0,0,
↦    0,-2,0,1,0,0,
↦    1,0,-2,0,1,0,
↦    0,1,0,-2,0,1,
↦    0,0,1,0,-2,1,
↦    0,0,0,1,1,-2
↦ [2]:
↦    0,0,0,0,0,0
↦ [3]:
↦    [1]:
↦       [1]:
↦          2,1,1
↦       [2]:
↦          4,1,1
↦    [2]:
↦       [1]:
↦          2,1,2
↦       [2]:
↦          4,1,2
↦    [3]:
↦       [1]:
↦          4,2,1
↦       [2]:
↦          6,2,1
↦    [4]:
↦       [1]:
↦          4,2,2
↦       [2]:
↦          6,2,2
↦    [5]:
↦       [1]:
↦          6,3,1
↦       [2]:
↦          7,3,1
↦    [6]:
↦       [1]:
↦          6,3,2
↦       [2]:
↦          7,3,2
↦ [4]:
↦    1,1,1,1,1,1
// The output is a list whose first entry contains the intersection matrix
// of the exceptional divisors. The second entry is the list of genera
// of these divisors. The third and fourth entry contain the information
// how to find the corresponding divisors in the respective charts.
```

A.5 Invariant Theory

A.5.1 G_a -Invariants

We work in characteristic 0 and use the Lie algebra generated by one vectorfield of the form $\sum x_i \partial / \partial x_{i+1}$.

```
  LIB "ainvar.lib";
  int n=5;
  int i;
  ring s=32003,(x(1..n)),wp(1,2,3,4,5);
  // definition of the vectorfield m=sum m[i,1]*d/dx(i)
  matrix m[n][1];
  for (i=1;i<=n-1;i=i+1)
  {
     m[i+1,1]=x(i);
  }
  // computation of the ring of invariants
  ideal in=invariantRing(m,x(2),x(1),0);
  in;   //invariant ring is generated by 5 invariants
↦ in[1]=x(1)
↦ in[2]=x(2)^2-2*x(1)*x(3)
↦ in[3]=x(3)^2-2*x(2)*x(4)+2*x(1)*x(5)
↦ in[4]=x(2)^3-3*x(1)*x(2)*x(3)+3*x(1)^2*x(4)
↦ in[5]=x(3)^3-3*x(2)*x(3)*x(4)-15997*x(1)*x(4)^2+3*x(2)^2*x(5)-6*x(1)*x(3)\
   *x(5)
  ring q=32003,(x,y,z,u,v,w),dp;
  matrix m[6][1];
  m[2,1]=x;
  m[3,1]=y;
  m[5,1]=u;
  m[6,1]=v;
  // the vectorfield is: xd/dy+yd/dz+ud/dv+vd/dw
  ideal in=invariantRing(m,y,x,0);
  in; //invariant ring is generated by 6 invariants
↦ in[1]=x
↦ in[2]=u
↦ in[3]=v2-2uw
↦ in[4]=zu-yv+xw
↦ in[5]=yu-xv
↦ in[6]=y2-2xz
```

A.5.2 Invariants of a finite group

Two algorithms to compute the invariant ring are implemented in SINGULAR, `invariant_ring` and `invariant_ring_random`, both by Agnes E. Heydtmann (`agnes@math.uni-sb.de`).

Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (J.Symb.Comput. 25, No.6, 727-731, 1998). In the non-modular case secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo the primary invariants, mapping to invariants with the Reynolds operator and using those or their power products such that they are linearly independent modulo the primary invariants (see paper

"Some Algorithms in Invariant Theory of Finite Groups" by Kemper and Steel (In: Proceedings of the Euroconference in Essen 1997, Birkhäuser Prog. Math. 173, 267-285, 1999)). In the modular case they are generated according to "Calculating Invariant Rings of Finite Groups over Arbitrary Fields" by Kemper (J.Symb.Comput. 21, No.3, 351-366, 1996).

We calculate now an example from Sturmfels: "Algorithms in Invariant Theory 2.3.7":

```
  LIB "finvar.lib";
  ring R=0,(x,y,z),dp;
  matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
  // the group G is generated by A in Gl(3,Q);
  print(A);
↦ 0, 1,0,
↦ -1,0,0,
↦ 0, 0,-1
  print(A*A*A*A); // the fourth power of A is 1
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
  // Use the first method to compute the invariants of G:
  matrix B(1..3);
  B(1..3)=invariant_ring(A);
  // SINGULAR returns 2 matrices, the first containing
  // primary invariants and the second secondary
  // invariants, i.e., module generators over a Noetherian
  // normalization
  // the third result are the irreducible secondary invariants
  // if the Molien series was available
  print(B(1));
↦ z2,x2+y2,x2y2
  print(B(2));
↦ 1,xyz,x2z-y2z,x3y-xy3
  print(B(3));
↦ xyz,x2z-y2z,x3y-xy3
  // Use the second method,
  // with random numbers between -1 and 1:
  B(1..3)=invariant_ring_random(A,1);
  print(B(1..3));
↦ z2,x2+y2,x4+y4-z4
↦  1,xyz,x2z-y2z,x3y-xy3
↦  xyz,x2z-y2z,x3y-xy3
```

A.6 Geometric Invariant Theory

A.6.1 GIT-Fans

Dolgachev/Hu and Thaddeus assigned to an algebraic variety with the action of an algebraic group the GIT-fan, a polyhedral fan enumerating the GIT-quotients in the sense of Mumford. The case of the action of an algebraic torus H on an affine variety X has been treated by Berchtold/Hausen. Based on their construction, an algorithm to compute the GIT-fan in this setting has been proposed be Keicher. Note that this setting is essential for many applications, since the torus case can be used to investigate the GIT-variation of the action of a connected reductive group G. In many important examples, X is symmetric under the action of a fnite group which either is known directly from its geometry or can be computed. A prominent instance is the Deligne-Mumford compactification M06bar of the moduli space of 6-pointed stable curves of genus zero, which has a natural action of the symmetric group S6. The library gitfan.lib implements an efficient algorithm for computing GIT-fans, which makes use of symmetries. We have applied this algorithm to determine the Mori chamber decomposition of the cone of movable divisor classes of M06bar. Each cone is represented by a single integer. The algorithm relies on Groebner basis techniques, convex geometry and actions of finite symmetry groups. It demonstrates the strength of cross-boarder methods in computer algebra, and the efficiency of the algorithms implemented in all involved areas. The algorithm is also suitable for parallel computations.

As an example we address in the following the computation of the GIT-Fan of M05bar.

We first compute the GIT-fan using the single line command provided by the library:

```
LIB "gitfan.lib";
setcores(4);
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
  T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
  T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
  T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
  T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
  T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
  1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
  1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
  0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
  0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
  0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
fan GIT = GITfan(J,Q);
fVector(GIT);
```

The GIT-Fan can be computed using symmetries as follows:

```
LIB "gitfan.lib";
setcores(4);
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
  T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
  T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
  T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
  T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
  T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
  1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
  1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
  0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
  0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
  0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
fan GIT2 = GITfan(J,Q,simplexSymmetryGroup);
GIT2;
```

Although we provide a procedure to compute the orbit decomposition of the group action on the simplex of variables this is not fast in Singular. In the following we describe how to use GAP to obtain the orbit decomposition and then continue with this data in Singular. This is particularly useful for more complicated examples.

The file orbits.gp in the directory doc of the Singular source tree contains GAP code to do this computation. This result is provided in the file doc/simplexOrbitRepresentativesG25.sing.

The file doc/simplexSymmetryGroupG25.sing contains the symmetry group (which here is S5).

Moreover the file doc/elementsInTermsOfGeneratorsG25.sing contains a representation of the elements of the symmetry group in terms of generators.

```
LIB "gitfan.lib";
setcores(4);
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
  T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
  T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
  T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
  T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
  T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
  1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
  1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
  0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
  0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
  0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
intmat Qt = transpose(Q);
<"doc/simplexOrbitRepresentativesG25.sing";
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
<"doc/simplexSymmetryGroupG25.sing";
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
apply(listOfOrbitConeOrbits,size);
list listOfMinimalOrbitConeOrbits = minimalOrbitConeOrbits(listOfOrbitConeOrbits);
size(listOfMinimalOrbitConeOrbits);
cone mov = coneViaPoints(transpose(Q));
mov = canonicalizeCone(mov);
list OC =  listOfOrbitConeOrbits;
<"doc/elementsInTermsOfGeneratorsG25.sing";
list Asigmagens = groupActionOnQImage(generatorsG,Q);
list actionOnOrbitconeIndicesForGenerators = groupActionOnHashes(Asigmagens,OC);
list actionOnOrbitconeIndices;
for (int i =1; i<=size(elementsInTermsOfGenerators);i++)
{
  actionOnOrbitconeIndices[i]=evaluateProduct(actionOnOrbitconeIndicesForGenerators,
}
list OClist = OC[1];
for (i =2;i<=size(OC);i++)
{
OClist = OClist + OC[i];
}
list SigmaHashes = GITfanParallelSymmetric(OClist, Q, mov, actionOnOrbitconeIndices)
SigmaHashes;
fan Sigma = hashesToFan(SigmaHashes,OClist);
```

Note that the result is not the complete fan but only the fan generated by a minimal set of representatives of maximal cones for the group action (by the group generated by Asigmagens).

A.7 Non-commutative Algebra

A.7.1 Left and two-sided Groebner bases

For a set of polynomials (resp. vectors) `S` in a non-commutative G-algebra, Singular:Plural provides two algorithms for computing Groebner bases.

The command `std` computes a left Groebner basis of a left module, generated by the set `S` (see Section 7.3.26 [std (plural)], page 360). The command `twostd (plural)` computes a two-sided Groebner basis (which is in particular also a left Groebner basis) of a two-sided ideal, generated by the set `S` (see Section 7.3.29 [twostd (plural)], page 363).

In the example below, we consider a particular set `S` in the algebra $A := U(sl_2)$ with the degree reverse lexicographic ordering. We compute a left Groebner basis `L` of the left ideal generated by `S` and a two-sided Groebner basis `T` of the two-sided ideal generated by `S`.
Then, we read off the information on the vector space dimension of the factor modules `A/L` and `A/T` using the command `vdim` (see Section 7.3.30 [vdim (plural)], page 364).
Further on, we use the command `reduce` (see Section 7.3.23 [reduce (plural)], page 356) to compare the left ideals generated by `L` and `T`.

We set `option(redSB)` and `option(redTail)` to make Singular compute completely reduced minimal bases of ideals (see Section 5.1.111 [option], page 234 and Section 7.4.2 [Groebner bases in G-algebras], page 366 for definitions and further details).

For long running computations, it is always recommended to set `option(prot)` to make Singular display some information on the performed computations (see Section 5.1.111 [option], page 234 for an interpretation of the displayed symbols).

```
// ----- 1.  setting up the algebra
  ring R = 0,(e,f,h),dp;
  matrix D[3][3];
  D[1,2]=-h; D[1,3]=2*e; D[2,3]=-2*f;
  def A=nc_algebra(1,D); setring A;
// ----- equivalently, you may use the following:
//  LIB "ncalg.lib";
//  def A = makeUsl2();
//  setring A;
// ----- 2.  defining the set S
  ideal S = e^3, f^3, h^3 - 4*h;
  option(redSB);
  option(redTail);
  option(prot);  // let us activate the protocol
  ideal L = std(S);
↦ 3(2)s
↦ s
↦ s
↦ 5s
↦ s
↦ (4)s
↦ 4(5)(4)s
↦ (6)(5)(4)s
↦ 3(7)4(5)(4)(3)s
↦ 3(4)(3)4(2)s
↦ (3)(2)s
↦ 3(5)(4)4(2)5
↦ (S:5)-----
↦ product criterion:7 chain criterion:12
  L;
↦ L[1]=h3-4h
↦ L[2]=fh2-2fh
↦ L[3]=eh2+2eh
↦ L[4]=2efh-h2-2h
↦ L[5]=f3
↦ L[6]=e3
  vdim(L); // the vector space dimension of the module A/L
↦ 15
  option(noprot); // turn off the protocol
  ideal T = twostd(S);
  T;
↦ T[1]=h3-4h
↦ T[2]=fh2-2fh
↦ T[3]=eh2+2eh
↦ T[4]=f2h-2f2
↦ T[5]=2efh-h2-2h
↦ T[6]=e2h+2e2
↦ T[7]=f3
↦ T[8]=ef2-fh
↦ T[9]=e2f-eh-2e
↦ T[10]=e3
  vdim(T);  // the vector space dimension of the module A/T
↦ 10
  print(matrix(reduce(L,T)));  // reduce L with respect to T
↦ 0,0,0,0,0,0
  // as we see, L is included in the left ideal generated by T
  print(matrix(reduce(T,L)));  // reduce T with respect to L
↦ 0,0,0,f2h-2f2,0,e2h+2e2,0,ef2-fh,e2f-eh-2e,0
  // the non-zero elements belong to T only
  ideal LT = twostd(L); // the two-sided Groebner basis of L
  // LT and T coincide as left ideals:
  size(reduce(LT,T));
↦ 0
  size(reduce(T,LT));
↦ 0
```

A.7.2 Right Groebner bases and syzygies

Most of the Singular:Plural commands correspond to the *left-sided* computations, that is left Groebner bases, left syzygies, left resolutions and so on. However, the *right-sided* computations can be done, using the *left-sided* functionality and *opposite* algebras.

In the example below, we consider the algebra $A := U(sl_2)$ and a set of generators $I = \{e^2, f\}$.

We will compute a left Groebner basis `LI` and a left syzygy module `LS` of a left ideal, generated by the set I .

Then, we define the opposite algebra `Aop` of `A`, set it as a basering, and create opposite objects of already computed ones.

Further on, we compute a right Groebner basis `RI` and a right syzygy module `RS` of a right ideal, generated by the set I in A .

```
// ----- setting up the algebra:
LIB "ncalg.lib";
def A = makeUsl2();
setring A; A;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    e f h
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    fe=ef-h
↦ //    he=eh+2e
↦ //    hf=fh-2f
// ----- equivalently, you may use
// ring AA = 0,(e,f,h),dp;
// matrix D[3][3];
// D[1,2]=-h; D[1,3]=2*e; D[2,3]=-2*f;
// def A=nc_algebra(1,D); setring A;
option(redSB);
option(redTail);
matrix T;
// --- define a generating set
ideal   I = e2,f;
ideal  LI = std(I); // the left Groebner basis of I
LI;             // we see that I was not a Groebner basis
↦ LI[1]=f
↦ LI[2]=h2+h
↦ LI[3]=eh+e
↦ LI[4]=e2
module LS = syz(I); // the left syzygy module of I
print(LS);
↦ -ef-2h+6,-f3,                      -ef2-fh+4f,  -e2f2-4efh+16ef-6h2+42h-72\
   ,
↦ e3,       e2f2-6efh-6ef+6h2+18h+12,e3f-3e2h-6e2,e4f
 // check: LS is a left syzygy, if T=0:
T  = transpose(LS)*transpose(I);
print(T);
↦ 0,
↦ 0,
↦ 0,
↦ 0
// --- let us define the opposite algebra of A
def Aop = opposite(A);
setring Aop; Aop;         // see how Aop looks like
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering a
↦ //                  : names    H F E
↦ //                  : weights  1 1 1
↦ //        block   2 : ordering ls
↦ //                  : names    H F E
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    FH=HF-2F
↦ //    EH=HE+2E
↦ //    EF=FE-H
// --- we "oppose" (transfer) objects from A to Aop
ideal   Iop = oppose(A,I);
ideal  RIop = std(Iop);  // the left Groebner basis of Iop in Aop
module RSop = syz(Iop);  // the left syzygy module of Iop in Aop
module LSop = oppose(A,LS);
module RLS  = syz(transpose(LSop));
// RLS is the left syzygy of transposed LSop in Aop
// --- let us return to A and transfer (i.e. oppose)
// all the computed objects back
setring A;
ideal  RI = oppose(Aop,RIop); // the right Groebner basis of I
RI;              // it differs from the left Groebner basis LI
↦ RI[1]=f
↦ RI[2]=h2-h
↦ RI[3]=eh+e
↦ RI[4]=e2
module RS = oppose(Aop,RSop); // the right syzygy module of I
print(RS);
↦ -ef+3h+6,-f3,                    -ef2+3fh,-e2f2+4efh+4ef,
↦ e3,      e2f2+2efh-6ef+2h2-10h+12,e3f,     e4f
 // check: RS is a right syzygy, if T=0:
T = matrix(I)*RS;
T;
↦ T[1,1]=0
↦ T[1,2]=0
↦ T[1,3]=0
↦ T[1,4]=0
module RLS;
RLS = transpose(oppose(Aop,RLS));
// RLS is the right syzygy of a left syzygy of I
// it is I itself ?
print(RLS);
↦ e2,f
```

A.8 Applications

A.8.1 Solving systems of polynomial equations

Here we turn our attention to the probably most popular aspect of the solving problem: given a system of complex polynomial equations with only finitely many solutions, compute floating point approximations for these solutions. This is widely considered as a task for numerical analysis. However, due to rounding errors, purely numerical methods are often unstable in an unpredictable way.

Therefore, in many cases, it is worth investing more computing power to derive additional knowledge on the geometric structure of the set of solutions (not to mention the question of how to decide whether the set of solutions is finite or not). The symbolic-numerical approach to the solving problem combines numerical methods with a symbolic preprocessing.

Depending on whether we want to preserve the multiplicities of the solutions or not, possible goals for a symbolic preprocessing are

- to find another system of generators (for instance, a reduced Groebner basis) for the ideal I generated by the polynomial equations. Alternatively, find a system of polynomials defining an ideal which has the same radical as I (see Section A.2 [Computing Groebner and Standard Bases], page 709, resp. Section D.4.26.12 [radical], page 1246).

In any case, the goal should be to find a system for which a numerical solution can be found more easily and in a more stable way. For systems with a large number of generators, the first step in a Singular computation could be to reduce the number of generators by applying the `interred` command (see Section 5.1.64 [interred], page 201). Another goal might be

- to decompose the system into several smaller (or, at least, more accessible) systems of polynomial equations. Then, the set of solutions of the original system is obtained by taking the union of the sets of solutions of the new systems.

Such a decomposition can be obtained in several ways: for instance, by computing a triangular decomposition (see Section D.8.5 [triang_lib], page 1855) for the ideal I, or by applying the factorizing Buchberger algorithm (see Section 5.1.34 [facstd], page 179), or by computing a primary decomposition of I (see Section D.4.26 [primdec_lib], page 1239).

Moreover, the equational modelling of a problem frequently causes unwanted solutions, for instance, zero as a multiple solution. Not only for stability reasons, one is frequently interested to get rid of those. This can be done by computing the saturation of I with respect to an ideal having the excess components as set of solutions (see Section D.4.7.7 [sat], page 1065).

The Singular libraries `solve.lib` and `triang.lib` provide several commands for solving systems of polynomial equations (based on a symbolic-numerical approach via Groebner bases, resp. resultants). In the example below, we show some of these commands at work.

```
LIB "solve.lib";
ring r=0,x(1..5),dp;
poly f0= x(1)^3+x(2)^2+x(3)^2+x(4)^2-x(5)^2;
poly f1= x(2)^3+x(1)^2+x(3)^2+x(4)^2-x(5)^2;
poly f2=x(3)^3+x(1)^2+x(2)^2+x(4)^2-x(5)^2;
poly f3=x(4)^2+x(1)^2+x(2)^2+x(3)^2-x(5)^2;
poly f4=x(5)^2+x(1)^2+x(2)^2+x(3)^2;
ideal i=f0,f1,f2,f3,f4;
ideal si=std(i);
//
// dimension of a solution set (here: 0) can be read from a Groebner bases
// (with respect to any global monomial ordering)
dim(si);
↦ 0
//
// the number of complex solutions (counted with multiplicities) is:
vdim(si);
↦ 108
//
// The given system has a multiple solution at the origin. We use facstd
// to compute equations for the non-zero solutions:
```

```
option(redSB);
ideal maxI=maxideal(1);
ideal j=sat(si,maxI);     // output is Groebner basis
vdim(j);                  // number of non-zero solutions (with mult's)
↦ 76
//
// We compute a triangular decomposition for the ideal I. This requires first
// the computation of a lexicographic Groebner basis (we use the FGLM
// conversion algorithm):
ring R=0,x(1..5),lp;
ideal j=fglm(r,j);
list L=triangMH(j);
size(L);                 // number of triangular components
↦ 7
L[1];                    // the first component
↦ _[1]=x(5)^2+1
↦ _[2]=x(4)^2+2
↦ _[3]=x(3)-1
↦ _[4]=x(2)^2
↦ _[5]=x(1)^2
//
// We compute floating point approximations for the solutions (with 30 digits)
def S=triang_solve(L,30);
↦
↦ // 'triang_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring S;
size(rlist);             // number of different non-zero solutions
↦ 28
rlist[1];                // the first solution
↦ [1]:
↦ 0
↦ [2]:
↦ 0
↦ [3]:
↦ 1
↦ [4]:
↦ (-I*1.41421356237309504880168872421)
↦ [5]:
↦ -I
//
// Alternatively, we could have applied directly the solve command:
setring r;
def T=solve(i,30,1,"nodisplay");  // compute all solutions with mult's
↦
↦ // 'solve' created a ring, in which a list SOL of numbers (the complex so\
   lutions)
↦ // is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; SOL;
setring T;
size(SOL);                // number of different solutions
↦ 4
SOL[1][1]; SOL[1][2];     // first solution and its multiplicity
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦    [3]:
↦ 1
↦    [4]:
↦ (i*2.44948974278317809819728407471)
↦    [5]:
↦ (i*1.73205080756887729352744634151)
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦    [3]:
↦ 1
↦    [4]:
↦ (-i*2.44948974278317809819728407471)
↦    [5]:
↦ (i*1.73205080756887729352744634151)
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦    [3]:
↦ 1
↦    [4]:
↦ (i*2.44948974278317809819728407471)
↦    [5]:
↦ (-i*1.73205080756887729352744634151)
↦ [4]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦    [3]:
↦ 1
↦    [4]:
↦ (-i*2.44948974278317809819728407471)
↦    [5]:
↦ (-i*1.73205080756887729352744634151)
↦ 1
SOL[size(SOL)];           // solutions of highest multiplicity
```

```
↦ [1]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦ 0
↦       [3]:
↦ 0
↦       [4]:
↦ 0
↦       [5]:
↦ 0
↦ [2]:
↦    32
//
// Or, we could remove the multiplicities first, by computing the
// radical:
setring r;
ideal k=std(radical(i));
vdim(k);                 // number of different complex solutions
↦ 29
def T1=solve(k,30,"nodisplay");  // compute all solutions with mult's
↦
↦ // 'solve' created a ring, in which a list SOL of numbers (the complex so\
   lutions)
↦ // is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; SOL;
setring T1;
size(SOL);               // number of different solutions
↦ 29
SOL[1];
↦ [1]:
↦ 1
↦ [2]:
↦ 1
↦ [3]:
↦ 1
↦ [4]:
↦ (-i*2.44948974278317809819728407471)
↦ [5]:
↦ (-i*1.73205080756887729352744634151)
```

A.8.2 AG codes

The library brnoeth.lib provides an implementation of the Brill-Noether algorithm for solving the Riemann-Roch problem and applications to Algebraic Geometry codes. The procedures can be applied to plane (singular) curves defined over a prime field of positive characteristic.

```
LIB "brnoeth.lib";
ring s=2,(x,y),lp;           // characteristic 2
poly f=x3y+y3+x;             // the Klein quartic
```

```
list KLEIN=Adj_div(f);        // compute the conductor
↦ Computing affine singular points ...
↦ Computing all points at infinity ...
↦ Computing affine singular places ...
↦ Computing singular places at infinity ...
↦ Computing non-singular places at infinity ...
↦ Adjunction divisor computed successfully
↦
↦ The genus of the curve is 3
KLEIN=NSplaces(1..3,KLEIN);   // computes places up to degree 3
↦ Computing non-singular affine places of degree 1 ...
↦ Computing non-singular affine places of degree 2 ...
↦ Computing non-singular affine places of degree 3 ...
KLEIN=extcurve(3,KLEIN);      // construct Klein quartic over F_8
↦
↦ Total number of rational places : NrRatPl = 24
↦
KLEIN[3];                     // display places (degree, number)
↦ [1]:
↦    1,1
↦ [2]:
↦    1,2
↦ [3]:
↦    1,3
↦ [4]:
↦    2,1
↦ [5]:
↦    3,1
↦ [6]:
↦    3,2
↦ [7]:
↦    3,3
↦ [8]:
↦    3,4
↦ [9]:
↦    3,5
↦ [10]:
↦    3,6
↦ [11]:
↦    3,7
// We define a divisor G of degree 14=6*1+4*2:
intvec G=6,0,0,4,0,0,0,0,0,0,0;   // 6 * place #1 + 4 * place #4
// We compute an evaluation code which evaluates at all rational places
// outside the support of G (place #4 is not rational)
intvec D=2..24;
// in D, the number i refers to the i-th element of the list POINTS in
// the ring KLEIN[1][5].
def RR=KLEIN[1][5];
setring RR; POINTS[1];        // the place in the support of G (not in supp(D))
↦ [1]:
↦ 0
↦ [2]:
↦ 1
↦ [3]:
↦ 0
setring s;
def RR=KLEIN[1][4];
↦ // ** redefining RR (def RR=KLEIN[1][4];) ./examples/AG_codes.sing:18
setring RR;
matrix C=AGcode_L(G,D,KLEIN); // generator matrix for the evaluation AG code
↦ Forms of degree 5 :
↦ 21
↦
↦ Vector basis successfully computed
↦
nrows(C);
↦ 12
ncols(C);
↦ 23
//
// We can also compute a generator matrix for the residual AG code
matrix CO=AGcode_Omega(G,D,KLEIN);
↦ Forms of degree 5 :
↦ 21
↦
↦ Vector basis successfully computed
↦
//
// Preparation for decoding:
// We need a divisor of degree at least 6 whose support is disjoint with the
// support of D:
intvec F=6;                   // F = 6*point #1
// in F, the i-th entry refers to the i-th element of the list POINTS in
// the ring KLEIN[1][5]
list K=prepSV(G,D,F,KLEIN);
↦ Forms of degree 5 :
↦ 21
↦
↦ Vector basis successfully computed
↦
↦ Forms of degree 4 :
↦ 15
↦
↦ Vector basis successfully computed
↦
↦ Forms of degree 4 :
↦ 15
↦
↦ Vector basis successfully computed
↦
K[size(K)][1];                // error-correcting capacity
↦ 3
//
//  Encoding and Decoding:
matrix word[1][11];           // a word of length 11 is encoded
word = 1,1,1,1,1,1,1,1,1,1,1;
```

```
def y=word*C0;                // the code word (length: 23)
matrix disturb[1][23];
disturb[1,1]=1;
disturb[1,10]=a;
disturb[1,12]=1+a;
y=y+disturb;                  // disturb the code word (3 errors)
def yy=decodeSV(y,K);         // error correction
yy-y;                         // display the error
↦ _[1,1]=1
↦ _[1,2]=0
↦ _[1,3]=0
↦ _[1,4]=0
↦ _[1,5]=0
↦ _[1,6]=0
↦ _[1,7]=0
↦ _[1,8]=0
↦ _[1,9]=0
↦ _[1,10]=(a)
↦ _[1,11]=0
↦ _[1,12]=(a+1)
↦ _[1,13]=0
↦ _[1,14]=0
↦ _[1,15]=0
↦ _[1,16]=0
↦ _[1,17]=0
↦ _[1,18]=0
↦ _[1,19]=0
↦ _[1,20]=0
↦ _[1,21]=0
↦ _[1,22]=0
↦ _[1,23]=0
```

Appendix B Polynomial data

B.1 Representation of mathematical objects

SINGULAR distinguishes between objects which do not belong to a ring and those which belong to a specific ring (see Section 3.3 [Rings and orderings], page 30). We comment only on the latter ones.

Internally all ring-dependent objects are polynomials or structures built from polynomials (and some additional information). Note that SINGULAR stores (and hence prints) a polynomial automatically w.r.t. the monomial ordering.

The definition of ideals and matrices, respectively, is straight forward: The user gives a list of polynomials which generate the ideal, resp. which are the entries of the matrix. (The number of rows and columns need to be provided when creating the matrix.)

A vector in SINGULAR is always an element of a free module over the basering. It is given as a list of polynomials in one of the following formats $[f_1,...,f_n]$ or $f_1*gen(1)+...+f_n*gen(n)$, where $gen(i)$ denotes the i-th canonical generator of a free module (with 1 at index i and 0 everywhere else). Both forms are equivalent. A vector is internally represented in the second form with the $gen(i)$ being "special" ring variables, ordered accordingly to the monomial ordering. Therefore, the form $[f_1,...,f_n]$ serves as output only if the monomial ordering gives priority to the component, i.e., is of the form `(c,...)` (see Section B.2.5 [Module orderings], page 769). However, in any case the procedure `show` from the library `inout.lib` displays the bracket format.

A vector $v=[f_1,...,f_n]$ should always be considered as a column vector in a free module of rank equal to nrows(v) where nrows(v) is equal to the maximal index r such that $f_r\neq 0$. This is due to the fact, that internally v is a polynomial in a sparse representation, i.e., $f_i*gen(i)$ is not stored if $f_i=0$ (for reasons of efficiency), hence the last 0-entries of v are lost. Only more complex structures are able to keep the rank.

A module M in SINGULAR is given by a list of vectors $v_1,...,v_k$ which generate the module as a submodule of the free module of rank equal to nrows(M) which is the maximum of nrows(v_i).

If one wants to create a module with a larger rank than given by its generators, one has to use the command `attrib(M,"rank",r)` (see Section 5.1.2 [attrib], page 156, Section 5.1.107 [nrows], page 232) or to define a matrix first, then converting it into a module. Modules in SINGULAR are almost the same as matrices, they may be considered as sparse representations of matrices. A module of a matrix is generated by the columns of the matrix and a matrix of a module has as columns the generators of the module. These conversions preserve the rank and the number of generators, resp. the number of rows and columns.

By the above remarks it might appear that SINGULAR is only able to handle submodules of a free module. However, this is not true. SINGULAR can compute with any finitely generated module over the basering R. Such a module, say N, is not represented by its generators but by its (generators and) relations. This means that $N=R^n/M$ where n is the number of generators of N and $M\subseteq R^n$ is the module of relations. In other words, defining a module M as a submodule of a free module R^n can also be considered as the definition of $N=R^n/M$.

Note that most functions, when applied to a module M, really deal with M. However, there are some functions which deal with $N=R^n/M$ instead of M.

For example, `std(M)` computes a standard basis of M (and thus gives another representation of N as $N=R^n/\mathrm{std}(M)$). However, `dim(M)`, resp. `vdim(M)`, return $\dim(R^n/M)$, resp. $\dim_k(R^n/M)$ (if M is given by a standard basis).

The function `syz(M)` returns the first syzygy module of M, i.e., the module of relations of the given generators of M which is equal to the second syzygy module of N. Refer to the description of each function in Section 5.1 [Functions], page 156 to get information which module the function deals with.

The numbering in `res` and other commands for computing resolutions refers to a resolution of $N = R^n/M$ (see Section 5.1.134 [res], page 253; Section C.3 [Syzygies and resolutions], page 775).

It is possible to compute in any field which is a valid ground field in SINGULAR. For doing so, one has to define a ring with the desired ground field and at least one variable. The elements of the field are of type number, but may also be considered as polynomials (of degree 0). Large computations should be faster if the elements of the field are defined as numbers.

The above remarks do also apply to quotient rings. Polynomial data are stored internally in the same manner, the only difference is that this polynomial representation is in general not unique. `reduce(f,std(0))` computes a normal form of a polynomial f in a quotient ring (cf. Section 5.1.131 [reduce], page 251).

B.2 Monomial orderings

B.2.1 Introduction to orderings

SINGULAR offers a great variety of monomial orderings which provide an enormous functionality, if used diligently. However, this flexibility might also be confusing for the novice user. Therefore, we recommend to those not familiar with monomial orderings to generally use the ordering `dp` for computations in the polynomial ring $K[x_1,\ldots,x_n]$, resp. `ds` for computations in the localization $\mathrm{Loc}_{(x)}K[x_1,\ldots,x_n]$.

For inhomogeneous input ideals, standard (resp. groebner) bases computations are generally faster with the orderings $\mathrm{Wp}(w_1,\ldots,w_n)$ (resp. $\mathrm{Ws}(w_1,\ldots,w_n)$) if the input is quasihomogenous w.r.t. the weights w_1, ..., w_n of x_1, ..., x_n.

If the output needs to be "triangular" (resp. "block-triangular"), the lexicographical ordering `lp` (resp. lexicographical block-orderings) need to be used. However, these orderings usually result in much less efficient computations.

B.2.2 General definitions for orderings

A monomial ordering (term ordering) on $K[x_1,\ldots,x_n]$ is a total ordering $<$ on the set of monomials (power products) $\{x^\alpha \mid \alpha \in \mathbf{N^n}\}$ which is compatible with the natural semigroup structure, i.e., $x^\alpha < x^\beta$ implies $x^\gamma x^\alpha < x^\gamma x^\beta$ for any $\gamma \in \mathbf{N^n}$. We do not require $<$ to be a wellordering. See the literature cited in Section C.9 [References], page 791.

It is known that any monomial ordering can be represented by a matrix M in $GL(n,R)$, but, of course, only integer coefficients are of relevance in practice.

Global orderings are wellorderings (i.e., $1 < x_i$ for each variable x_i), local orderings satisfy $1 > x_i$ for each variable. If some variables are ordered globally and others locally we call it a mixed ordering. Local or mixed orderings are not wellorderings.

Let K be the ground field, $x = (x_1,\ldots,x_n)$ the variables and $<$ a monomial ordering, then Loc $K[x]$ denotes the localization of $K[x]$ with respect to the multiplicatively closed set

$$\{1+g \mid g=0 \text{ or } g \in K[x]\backslash\{0\} \text{ and } L(g)<1\}.$$

Here, $L(g)$ denotes the leading monomial of g, i.e., the biggest monomial of g with respect to $<$. The result of any computation which uses standard basis computations has to be interpreted in Loc $K[x]$.

Note that the definition of a ring includes the definition of its monomial ordering (see Section 3.3 [Rings and orderings], page 30). SINGULAR offers the monomial orderings described in the following sections.

B.2.3 Global orderings

For all these orderings, we have Loc $K[x] = K[x]$

lp: lexicographical ordering:
$x^\alpha < x^\beta \Leftrightarrow \exists\, 1 \le i \le n : \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}, \alpha_i < \beta_i$.

rp: inverse lexicographical ordering:
$x^\alpha < x^\beta \Leftrightarrow \exists\, 1 \le i \le n : \alpha_n = \beta_n, \ldots, \alpha_{i+1} = \beta_{i+1}, \alpha_i < \beta_i$.

dp: degree reverse lexicographical ordering:
let $\deg(x^\alpha) = \alpha_1 + \cdots + \alpha_n$, then $x^\alpha < x^\beta \Leftrightarrow \deg(x^\alpha) < \deg(x^\beta)$ or
$\deg(x^\alpha) = \deg(x^\beta)$ and $\exists\, 1 \le i \le n : \alpha_n = \beta_n, \ldots, \alpha_{i+1} = \beta_{i+1}, \alpha_i > \beta_i$.

Dp: degree lexicographical ordering:
let $\deg(x^\alpha) = \alpha_1 + \cdots + \alpha_n$, then $x^\alpha < x^\beta \Leftrightarrow \deg(x^\alpha) < \deg(x^\beta)$ or
$\deg(x^\alpha) = \deg(x^\beta)$ and $\exists\, 1 \le i \le n : \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}, \alpha_i < \beta_i$.

wp: weighted reverse lexicographical ordering:
let $w_1, \ldots, w_n$ be positive integers. Then `wp`$(w_1, \ldots, w_n)$ is defined as `dp` but with $\deg(x^\alpha) = w_1\alpha_1 + \cdots + w_n\alpha_n$.

Wp: weighted lexicographical ordering:
let $w_1, \ldots, w_n$ be positive integers. Then `Wp`$(w_1, \ldots, w_n)$ is defined as `Dp` but with $\deg(x^\alpha) = w_1\alpha_1 + \cdots + w_n\alpha_n$.

B.2.4 Local orderings

For ls, ds, Ds and, if the weights are positive integers, also for ws and Ws, we have Loc $K[x] = K[x]_{(x)}$, the localization of $K[x]$ at the maximal ideal $(x) = (x_1, ..., x_n)$.

ls: negative lexicographical ordering:
$x^\alpha < x^\beta \Leftrightarrow \exists\, 1 \le i \le n : \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}, \alpha_i > \beta_i$.

ds: negative degree reverse lexicographical ordering:
let $\deg(x^\alpha) = \alpha_1 + \cdots + \alpha_n$, then $x^\alpha < x^\beta \Leftrightarrow \deg(x^\alpha) > \deg(x^\beta)$ or
$\deg(x^\alpha) = \deg(x^\beta)$ and $\exists\, 1 \le i \le n : \alpha_n = \beta_n, \ldots, \alpha_{i+1} = \beta_{i+1}, \alpha_i > \beta_i$.

Ds: negative degree lexicographical ordering:
let $\deg(x^\alpha) = \alpha_1 + \cdots + \alpha_n$, then $x^\alpha < x^\beta \Leftrightarrow \deg(x^\alpha) > \deg(x^\beta)$ or
$\deg(x^\alpha) = \deg(x^\beta)$ and $\exists\, 1 \le i \le n : \alpha_1 = \beta_1, \ldots, \alpha_{i-1} = \beta_{i-1}, \alpha_i < \beta_i$.

ws: (general) weighted reverse lexicographical ordering:
`ws`$(w_1, \ldots, w_n)$, w_1 a nonzero integer, $w_2, \ldots, w_n$ any integer (including 0), is defined as `ds` but with $\deg(x^\alpha) = w_1\alpha_1 + \cdots + w_n\alpha_n$.

Ws: (general) weighted lexicographical ordering:
`Ws`$(w_1, \ldots, w_n)$, w_1 a nonzero integer, $w_2, \ldots, w_n$ any integer (including 0), is defined as `Ds` but with $\deg(x^\alpha) = w_1\alpha_1 + \cdots + w_n\alpha_n$.

B.2.5 Module orderings

Singular offers also orderings on the set of "monomials" $\{x^a e_i \mid a \in N^n, 1 \le i \le r\}$ in Loc $K[x]^r$ = Loc $K[x]e_1 + \ldots +$Loc $K[x]e_r$, where $e_1, \ldots, e_r$ denote the canonical generators of Loc $K[x]^r$, the r-fold direct sum of Loc $K[x]$. (The function `gen(i)` yields e_i).

We have two possibilities: either to give priority to the component of a vector in Loc $K[x]^r$ or (which is the default in Singular) to give priority to the coefficients. The orderings `(<,c)` and

`(<,C)` give priority to the coefficients; whereas `(c,<)` and `(C,<)` give priority to the components. Let `<` be any of the monomial orderings of Loc $K[x]$ as above.

(<,C): $<_m= (<, C)$ denotes the module ordering (giving priority to the coefficients):
$x^\alpha e_i <_m x^\beta e_j \Leftrightarrow x^\alpha < x^\beta$ or $(x^\alpha = x^\beta$ and $i < j)$.

Example:

```
  ring r = 0, (x,y,z), ds;
  // the same as ring r = 0, (x,y,z), (ds, C);
  [x+y2,z3+xy];
↦ x*gen(1)+xy*gen(2)+y2*gen(1)+z3*gen(2)
  [x,x,x];
↦ x*gen(3)+x*gen(2)+x*gen(1)
```

(C,<): $<_m= (C, <)$ denotes the module ordering (giving priority to the component):
$x^\alpha e_i <_m x^\beta e_j \Leftrightarrow i < j$ or $(i = j$ and $x^\alpha < x^\beta)$.

Example:

```
  ring r = 0, (x,y,z), (C,lp);
  [x+y2,z3+xy];
↦ xy*gen(2)+z3*gen(2)+x*gen(1)+y2*gen(1)
  [x,x,x];
↦ x*gen(3)+x*gen(2)+x*gen(1)
```

(<,c): $<_m= (<, c)$ denotes the module ordering (giving priority to the coefficients):
$x^\alpha e_i <_m x^\beta e_j \Leftrightarrow x^\alpha < x^\beta$ or $(x^\alpha = x^\beta$ and $i > j)$.

Example:

```
  ring r = 0, (x,y,z), (lp,c);
  [x+y2,z3+xy];
↦ xy*gen(2)+x*gen(1)+y2*gen(1)+z3*gen(2)
  [x,x,x];
↦ x*gen(1)+x*gen(2)+x*gen(3)
```

(c,<): $<_m= (c, <)$ denotes the module ordering (giving priority to the component):
$x^\alpha e_i <_m x^\beta e_j \Leftrightarrow i > j$ or $(i = j$ and $x^\alpha < x^\beta)$.

Example:

```
  ring r = 0, (x,y,z), (c,lp);
  [x+y2,z3+xy];
↦ [x+y2,xy+z3]
  [x,x,x];
↦ [x,x,x]
```

The output of a vector v in $K[x]^r$ with components $v_1, \ldots, v_r$ has the format $v_1 * gen(1) + \ldots + v_r * gen(r)$ (up to permutation) unless the ordering starts with `c`. In this case a vector is written as $[v_1, \ldots, v_r]$. In all cases Singular can read input in both formats.

B.2.6 Matrix orderings

Let M be an invertible $(n \times n)$-matrix with integer coefficients and $M_1, \ldots, M_n$ the rows of M.

The M-ordering `<` is defined as follows:
$x^a < x^b \Leftrightarrow \exists\ 1 \le i \le n : M_1 a = M_1 b, \ldots, M_{i-1} a = M_{i-1} b$ and $M_i a < M_i b$.

Thus, $x^a < x^b$ if and only if Ma is smaller than Mb with respect to the lexicographical ordering.

The following matrices represent (for 3 variables) the global and local orderings defined above (note that the matrix is not uniquely determined by the ordering):

$$\text{lp: } \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{dp: } \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{Dp: } \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\text{wp(1,2,3): } \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{Wp(1,2,3): } \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\text{ls: } \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \text{ds: } \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{Ds: } \begin{pmatrix} -1 & -1 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\text{ws(1,2,3): } \begin{pmatrix} -1 & -2 & -3 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \text{Ws(1,2,3): } \begin{pmatrix} -1 & -2 & -3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Product orderings (see next section) represented by a matrix:

$$\text{(dp(3), wp(1,2,3)): } \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\text{(Dp(3), ds(3)): } \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Orderings with extra weight vector (see below) represented by a matrix:

$$\text{(dp(3), a(1,2,3),dp(3)): } \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\text{(a(1,2,3,4,5),Dp(3), ds(3)): } \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix}$$

Example:

```
        ring r = 0, (x,y,z), M(1, 0, 0,   0, 1, 0,   0, 0, 1);
```

which may also be written as:

```
        intmat m[3][3]=1, 0, 0, 0, 1, 0, 0, 0, 1;
        m;
    ↦ 1,0,0,
    ↦ 0,1,0,
    ↦ 0,0,1
    ring r = 0, (x,y,z), M(m);
    r;
  ↦ // coefficients: QQ
  ↦ // number of vars : 3
  ↦ //        block   1 : ordering M
  ↦ //                  : names    x y z
  ↦ //                  : weights  1 0 0
  ↦ //                  : weights  0 1 0
  ↦ //                  : weights  0 0 1
  ↦ //        block   2 : ordering C
```

If the ring has n variables and the matrix does not contain $n \times n$ entries, an error message is given.

WARNING: Singular does not check whether the matrix has full rank. In such a case some computations might not terminate, others may not give a sensible result.

Having these matrix orderings Singular can compute standard bases for any monomial ordering which is compatible with the natural semigroup structure. In practice the global and local orderings together with block orderings should be sufficient in most cases. These orderings are faster than the corresponding matrix orderings, since evaluating a matrix product is time consuming.

B.2.7 Product orderings

Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ be two ordered sets of variables, $<_1$ a monomial ordering on $K[x]$ and $<_2$ a monomial ordering on $K[y]$. The product ordering (or block ordering) $< := (<_1 , <_2)$ on $K[x, y]$ is the following:
$x^a y^b < x^A y^B \Leftrightarrow x^a <_1 x^A$ or $(x^a = x^A$ and $y^b <_2 y^B)$.

Inductively one defines the product ordering of more than two monomial orderings.

In Singular, any of the above global orderings, local orderings or matrix orderings may be combined (in an arbitrary manner and length) to a product ordering. E.g., `(lp(3), M(1, 2, 3, 1, 1, 1, 1, 0, 0), ds(4), ws(1,2,3))` defines: `lp` on the first 3 variables, the matrix ordering `M(1, 2, 3, 1, 1, 1, 1, 0, 0)` on the next 3 variables, `ds` on the next 4 variables and `ws(1,2,3)` on the last 3 variables.

B.2.8 Extra weight vector

`a`$(w_1, \ldots, w_n)$, $w_1, \ldots, w_n$ any integers (including 0), defines $\deg(x^\alpha) = w_1\alpha_1 + \cdots + w_n\alpha_n$ and

$$\deg(x^\alpha) < \deg(x^\beta) \Rightarrow x^\alpha < x^\beta,$$

$$\deg(x^\alpha) > \deg(x^\beta) \Rightarrow x^\alpha > x^\beta.$$

An extra weight vector does not define a monomial ordering by itself: it can only be used in combination with other orderings to insert an extra line of weights into the ordering matrix.

Example:

```
ring r = 0, (x,y,z),  (a(1,2,3),wp(4,5,2));
ring s = 0, (x,y,z),  (a(1,2,3),dp);
ring q = 0, (a,b,c,d),(lp(1),a(1,2,3),ds);
```

B.2.9 Pseudo ordering L

L(max_exponent) is not an ordering but sets the maximal allowed exponent for polynomial in this ring. The default is 32767. The current value for a ring is reflected in the attribute `"maxExp"`. This attribute is also set (and acknowledged) for the list constructed by `ringlist` and the construction of a ring from such a list.

Appendix C Mathematical background

This chapter introduces some of the mathematical notions and definitions used throughout the manual. It is mostly a collection of the most prominent definitions and properties. For details, please, refer to articles or text books (see Section C.9 [References], page 791).

C.1 Standard bases

Definition

Let $R = \mathrm{Loc}_< K[\underline{x}]$ and let I be a submodule of R^r. Note that for r=1 this means that I is an ideal in R. Denote by $L(I)$ the submodule of R^r generated by the leading terms of elements of I, i.e. by $\{L(f) \mid f \in I\}$. Then $f_1, \ldots, f_s \in I$ is called a **standard basis** of I if $L(f_1), \ldots, L(f_s)$ generate $L(I)$.

A standard basis is **minimal** if $\forall i : (f_1, .., f_{i-1}, f_{i+1}, .., f_s) \neq I$.

A minimal standard basis is **completely reduced** if $\forall i :$ `reduce`$(f_i, (f_1, .., f_{i-1}, f_{i+1}, .., f_s)) = f_i$

Properties

normal form:

A function $\mathrm{NF} : R^r \times \{G \mid G \text{ a standard basis}\} \to R^r, (p, G) \mapsto \mathrm{NF}(p|G)$, is called a **normal form** if for any $p \in R^r$ and any standard basis G the following holds: if $\mathrm{NF}(p|G) \neq 0$ then $L(g)$ does not divide $L(\mathrm{NF}(p|G))$ for all $g \in G$. The function may also be applied to any generating set of an ideal: the result is then not uniquely defined.

$\mathrm{NF}(p|G)$ is called a **normal form of p with respect to G**

ideal membership:

For a standard basis G of I the following holds: $f \in I$ if and only if $\mathrm{NF}(f, G) = 0$.

Hilbert function:

Let $I \subseteq K[\underline{x}]^r$ be a homogeneous module, then the Hilbert function H_I of I (see below) and the Hilbert function $H_{L(I)}$ of the leading module $L(I)$ coincide, i.e., $H_I = H_{L(I)}$.

C.2 Hilbert function

Let M $= \bigoplus_{i \in Z} M_i$ be a graded module over $K[x_1, .., x_n]$ with respect to weights $(w_1, ..w_n)$. The **Hilbert function** of M, H_M, is defined (on the integers) by

$$H_M(k) := dim_K M_k.$$

The **Hilbert-Poincare series** of M is the power series

$$\mathrm{HP}_M(t) := \sum_{i=-\infty}^{\infty} H_M(i) t^i = \sum_{i=-\infty}^{\infty} dim_K M_i \cdot t^i.$$

It turns out that $\mathrm{HP}_M(t)$ can be written in two useful ways for weights $(1, .., 1)$:

$$\mathrm{HP}_M(t) = \frac{Q(t)}{(1-t)^n} = \frac{P(t)}{(1-t)^{dim(M)}}$$

where $Q(t)$ and $P(t)$ are polynomials in $\mathbf{Z}[t]$. $Q(t)$ is called the **first Hilbert series**, and $P(t)$ the **second Hilbert series**. If $P(t) = \sum_{k=0}^{N} a_k t^k$, and $d = dim(M)$, then $H_M(s) = \sum_{k=0}^{N} a_k \binom{d+s-k-1}{d-1}$ (the

Hilbert polynomial) for $s \geq N$.

Generalizing this to quasihomogeneous modules we get

$$\mathrm{HP}_M(t) = \frac{Q(t)}{\Pi_{i=1}^{n}(1-t^{w_i})}$$

where $Q(t)$ is a polynomial in $\mathbf{Z}[t]$. $Q(t)$ is called the **first (weighted) Hilbert series** of M.

C.3 Syzygies and resolutions

Syzygies

Let R be a quotient of $\mathrm{Loc}_{<}K[\underline{x}]$ and let $I=(g_1,...,g_s)$ be a submodule of R^r. Then the **module of syzygies** (or **1st syzygy module**, **module of relations**) of I, syz(I), is defined to be the kernel of the map $R^s \to R^r$, $\sum_{i=1}^{s} w_i e_i \mapsto \sum_{i=1}^{s} w_i g_i$.

The **k-th syzygy module** is defined inductively to be the module of syzygies of the $(k-1)$-st syzygy module.

Note, that the syzygy modules of I depend on a choice of generators $g_1,...,g_s$. But one can show that they depend on I uniquely up to direct summands.

Example:

```
  ring R= 0,(u,v,x,y,z),dp;
  ideal i=ux, vx, uy, vy;
  print(syz(i));
↦ -y,0, -v,0,
↦ 0, -y,u, 0,
↦ x, 0, 0, -v,
↦ 0, x, 0, u
```

Free resolutions

Let $I=(g_1,...,g_s) \subseteq R^r$ and $M=R^r/I$. A **free resolution of** M is a long exact sequence

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow M \longrightarrow 0,$$

where the columns of the matrix A_1 generate I . Note that resolutions need not to be finite (i.e., of finite length). The Hilbert Syzygy Theorem states that for $R = \mathrm{Loc}_{<}K[\underline{x}]$ there exists a ("minimal") resolution of length not exceeding the number of variables.

Example:

```
  ring R= 0,(u,v,x,y,z),dp;
  ideal I = ux, vx, uy, vy;
  resolution resI = mres(I,0); resI;
↦  1      4      4      1
↦ R <--  R <--  R <--  R
↦
↦ 0      1      2      3
↦
  // The matrix A_1 is given by
  print(matrix(resI[1]));
```

```
↦ vy,uy,vx,ux
  // We see that the columns of A_1 generate I.
  // The matrix A_2 is given by
  print(matrix(resI[3]));
↦ u,
↦ -v,
↦ -x,
↦ y
```

Betti numbers and regularity

Let R be a graded ring (e.g., $R = \mathrm{Loc}_< K[\underline{x}]$) and let $I \subset R^r$ be a graded submodule. Let

$$R^r = \bigoplus_a R \cdot e_{a,0} \xleftarrow{A_1} \bigoplus_a R \cdot e_{a,1} \longleftarrow \dots \longleftarrow \bigoplus_a R \cdot e_{a,n} \longleftarrow 0$$

be a minimal free resolution of R^r/I considered with homogeneous maps of degree 0. Then the **graded Betti number** $b_{i,j}$ of R^r/I is the minimal number of generators $e_{a,j}$ in degree $i+j$ of the j-th syzygy module of R^r/I (i.e., the $(j-1)$-st syzygy module of I). Note that, by definition, the 0-th syzygy module of R^r/I is R^r and the 1st syzygy module of R^r/I is I.

The **regularity** of I is the smallest integer s such that

$$\deg(e_{a,j}) \le s + j - 1 \quad \text{for all } j.$$

Example:

```
  ring R= 0,(u,v,x,y,z),dp;
  ideal I = ux, vx, uy, vy;
  resolution resI = mres(I,0); resI;
↦  1      4      4      1
↦ R <--  R <--  R <--  R
↦
↦ 0      1      2      3
↦
  // the betti number:
  print(betti(resI), "betti");
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     4     4     1
↦ ------------------------------
↦ total:     1     4     4     1
↦
  // the regularity:
  regularity(resI);
↦ 2
```

C.4 Characteristic sets

Let $<$ be the lexicographical ordering on $R = K[x_1, ..., x_n]$ with $x_1 < ... < x_n$. For $f \in R$ let lvar(f) (the leading variable of f) be the largest variable in f, i.e., if $f = a_s(x_1, ..., x_{k-1})x_k^s + ... + a_0(x_1, ..., x_{k-1})$ for some $k \le n$ then lvar$(f) = x_k$.

Moreover, let $\mathrm{ini}(f) := a_s(x_1, ..., x_{k-1})$. The pseudoremainder $r = \mathrm{prem}(g, f)$ of g with respect to f is defined by the equality $\mathrm{ini}(f)^a \cdot g = qf + r$ with $\deg_{lvar(f)}(r) < \deg_{lvar(f)}(f)$ and a minimal.

A set $T = \{f_1, ..., f_r\} \subset R$ is called triangular if $\mathrm{lvar}(f_1) < ... < \mathrm{lvar}(f_r)$. Moreover, let $U \subset T$, then (T, U) is called a triangular system, if T is a triangular set such that $\mathrm{ini}(T)$ does not vanish on $V(T) \setminus V(U)(=: V(T \setminus U))$.

T is called irreducible if for every i there are no d_i,f'_i,f''_i such that

$$\mathrm{lvar}(d_i) < \mathrm{lvar}(f_i) = \mathrm{lvar}(f'_i) = \mathrm{lvar}(f''_i),$$

$$0 \notin \mathrm{prem}(\{d_i, \mathrm{ini}(f'_i), \mathrm{ini}(f''_i)\}, \{f_1, ..., f_{i-1}\}),$$

$$\mathrm{prem}(d_i f_i - f'_i f''_i, \{f_1, ..., f_{i-1}\}) = 0.$$

Furthermore, (T, U) is called irreducible if T is irreducible.

The main result on triangular sets is the following: Let $G = \{g_1, ..., g_s\} \subset R$, then there are irreducible triangular sets $T_1, ..., T_l$ such that $V(G) = \bigcup_{i=1}^{l}(V(T_i \setminus I_i))$ where $I_i = \{\mathrm{ini}(f) \mid f \in T_i\}$. Such a set $\{T_1, ..., T_l\}$ is called an **irreducible characteristic series** of the ideal (G).

Example:

```
  ring R= 0,(x,y,z,u),dp;
  ideal i=-3zu+y2-2x+2,
          -3x2u-4yz-6xz+2y2+3xy,
          -3z2u-xu+y2z+y;
  print(char_series(i));
↦ _[1,1],3x2z-y2+2yz,3x2u-3xy-2y2+2yu,
↦ x,     -y+2z,      -2y2+3yu-4
```

C.5 Gauss-Manin connection

Let $f\colon (C^{n+1}, 0) \to (C, 0)$ be a complex isolated hypersurface singularity given by a polynomial with algebraic coefficients which we also denote by f. Let $O = C[x_0, \ldots, x_n]_{(x_0,\ldots,x_n)}$ be the local ring at the origin and J_f the Jacobian ideal of f.

A **Milnor representative** of f defines a differentiable fibre bundle over the punctured disc with fibres of homotopy type of μ n-spheres. The n-th cohomology bundle is a flat vector bundle of dimension n and carries a natural flat connection with covariant derivative ∂_t. The **monodromy operator** is the action of a positively oriented generator of the fundamental group of the punctured disc on the Milnor fibre. Sections in the cohomology bundle of **moderate growth** at 0 form a regular $D = C\{t\}[\partial_t]$-module G, the **Gauss-Manin connection**.

By integrating along flat multivalued families of cycles, one can consider fibrewise global holomorphic differential forms as elements of G. This factors through an inclusion of the **Brieskorn lattice** $H'' := \Omega^{n+1}_{C^{n+1},0}/df \wedge d\Omega^{n-1}_{C^{n+1},0}$ in G.

The D-module structure defines the **V-filtration** V on G by $V^\alpha := \sum_{\beta \geq \alpha} C\{t\}ker(t\partial_t - \beta)^{n+1}$. The Brieskorn lattice defines the **Hodge filtration** F on G by $F_k = \partial_t^k H''$ which comes from the **mixed Hodge structure** on the Milnor fibre. Note that $F_{-1} = H'$.

The induced V-filtration on the Brieskorn lattice determines the **singularity spectrum** Sp by $Sp(\alpha) := \dim_C Gr_V^\alpha Gr_0^F G$. The spectrum consists of μ rational numbers $\alpha_1, \ldots, \alpha_\mu$ such that $e^{2\pi i\alpha_1}, \ldots, e^{2\pi i\alpha_\mu}$ are the eigenvalues of the monodromy. These **spectral numbers** lie in the open interval $(-1, n)$, symmetric about the midpoint $(n-1)/2$.

The spectrum is constant under μ-constant deformations and has the following semicontinuity property: The number of spectral numbers in an interval $(a, a+1]$ of all singularities of a small deformation of f is greater than or equal to that of f in this interval. For semiquasihomogeneous singularities, this also holds for intervals of the form $(a, a+1)$.

Two given isolated singularities f and g determine two spectra and from these spectra we get an integer. This integer is the maximal positive integer k such that the semicontinuity holds for the spectrum of f and k times the spectrum of g. These numbers give bounds for the maximal number of isolated singularities of a specific type on a hypersurface $X \subset P^n$ of degree d: such a hypersurface has a smooth hyperplane section, and the complement is a small deformation of a cone over this hyperplane section. The cone itself being a μ-constant deformation of $x_0^d + \ldots + x_n^d = 0$, the singularities are bounded by the spectrum of $x_0^d + \ldots + x_n^d$.

Using the library `gmssing.lib` one can compute the **monodromy**, the V-fitration on H''/H', and the spectrum.

Let us consider as an example $f = x^5 + x^2y^2 + y^5$. First, we compute a matrix M such that $\exp(2\pi i M)$ is a monodromy matrix of f and the Jordan normal form of M :

```
  LIB "mondromy.lib";
  ring R=0,(x,y),ds;
  poly f=x5+x2y2+y5;
  matrix M=monodromyB(f);
  print(M);
↦ 11/10,0,    0,    0,    0,   0,-1/4,0,   0,   0,  0,
↦ 0,    13/10,0,    0,    0,   0,0,   15/8,0,   0,  0,
↦ 0,    0,    13/10,0,    0,   0,0,   0,   15/8,0,  0,
↦ 0,    0,    0,    11/10,-1/4,0,0,   0,   0,   0,  0,
↦ 0,    0,    0,    0,    9/10,0,0,   0,   0,   0,  0,
↦ 0,    0,    0,    0,    0,   1,0,   0,   0,   0,  3/5,
↦ 0,    0,    0,    0,    0,   0,9/10,0,   0,   0,  0,
↦ 0,    0,    0,    0,    0,   0,0,   7/10,0,   0,  0,
↦ 0,    0,    0,    0,    0,   0,0,   0,   7/10,0,  0,
↦ 0,    0,    0,    0,    0,   0,0,   0,   1,   -2/5,
↦ 0,    0,    0,    0,    0,   0,0,   0,   5/8,0
```

Now, we compute the V-fitration on H''/H' and the spectrum:

```
  LIB "gmssing.lib";
  ring R=0,(x,y),ds;
  poly f=x5+x2y2+y5;
  list l=vfilt(f);
  print(l[1]); // spectral numbers
↦ -1/2,
↦ -3/10,
↦ -1/10,
↦ 0,
↦ 1/10,
↦ 3/10,
↦ 1/2
  print(l[2]); // corresponding multiplicities
↦ 1,
↦ 2,
↦ 2,
↦ 1,
↦ 2,
↦ 2,
↦ 1
  print(l[3]); // vector space of i-th graded part
↦ [1]:
↦    _[1]=gen(11)
↦ [2]:
↦    _[1]=gen(10)
↦    _[2]=gen(6)
↦ [3]:
↦    _[1]=gen(9)
↦    _[2]=gen(4)
↦ [4]:
↦    _[1]=gen(5)
↦ [5]:
↦    _[1]=gen(3)
↦    _[2]=gen(8)
↦ [6]:
↦    _[1]=gen(2)
↦    _[2]=gen(7)
↦ [7]:
↦    _[1]=gen(1)
  print(l[4]); // monomial vector space basis of H''/s*H''
↦ y5,
↦ y4,
↦ y3,
↦ y2,
↦ xy,
↦ y,
↦ x4,
↦ x3,
↦ x2,
↦ x,
↦ 1
  print(l[5]); // standard basis of Jacobian ideal
↦ 2x2y+5y4,
↦ 5x5-5y5,
↦ 2xy2+5x4,
↦ 10y6+25x3y4
```

Here `l[1]` contains the spectral numbers, `l[2]` the corresponding multiplicities, `l[3]` a C -basis of the V-filtration on H''/H' in terms of the monomial basis of $O/J_f \cong H''/H'$ in `l[4]` (separated by degree).

If the principal part of f is C-nondegenerate, one can compute the spectrum using the library `spectrum.lib`. In this case, the V-filtration on H'' coincides with the Newton-filtration on H'' which allows to compute the spectrum more efficiently.

Let us calculate one specific example, the maximal number of triple points of type $\tilde{E}_6$ on a surface $X \subset P^3$ of degree seven. This calculation can be done over the rationals. We choose a local ordering on $Q[x, y, z]$. Here we take the negative degree lexicographical ordering, in Singular denoted by `ds`:

```
ring r=0,(x,y,z),ds;
LIB "spectrum.lib";
poly f=x^7+y^7+z^7;
list s1=spectrumnd( f );
s1;
↦ [1]:
↦    _[1]=-4/7
↦    _[2]=-3/7
↦    _[3]=-2/7
↦     _[4]=-1/7
↦     _[5]=0
↦     _[6]=1/7
↦     _[7]=2/7
↦     _[8]=3/7
↦     _[9]=4/7
↦     _[10]=5/7
↦     _[11]=6/7
↦     _[12]=1
↦     _[13]=8/7
↦     _[14]=9/7
↦     _[15]=10/7
↦     _[16]=11/7
↦  [2]:
↦     1,3,6,10,15,21,25,27,27,25,21,15,10,6,3,1
```

The command `spectrumnd(f)` computes the spectrum of f and returns a list with six entries: The Milnor number $\mu(f)$, the geometric genus $p_g(f)$ and the number of different spectrum numbers. The other three entries are of type `intvec`. They contain the numerators, denominators and multiplicities of the spectrum numbers. So $x^7+y^7+z^7=0$ has Milnor number 216 and geometrical genus 35. Its spectrum consists of the 16 different rationals
$\frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, \frac{1}{1}, \frac{8}{7}, \frac{9}{7}, \frac{10}{7}, \frac{11}{7}, \frac{12}{7}, \frac{13}{7}, \frac{2}{1}, \frac{15}{7}, \frac{16}{7}, \frac{17}{7}, \frac{18}{7}$
appearing with multiplicities
1,3,6,10,15,21,25,27,27,25,21,15,10,6,3,1.

The singularities of type $\tilde{E}_6$ form a μ-constant one parameter family given by $x^3+y^3+z^3+\lambda xyz=0, \quad \lambda^3 \neq -27$. Therefore they have all the same spectrum, which we compute for $x^3+y^3+z^3$.

```
poly g=x^3+y^3+z^3;
list s2=spectrumnd(g);
s2;
↦  [1]:
↦     8
↦  [2]:
↦     1
↦  [3]:
↦     4
↦  [4]:
↦     1,4,5,2
↦  [5]:
↦     1,3,3,1
↦  [6]:
↦     1,3,3,1
```

Evaluating semicontinuity is very easy:

```
semicont(s1,s2);
↦  18
```

This tells us that there are at most 18 singularities of type $\tilde{E}_6$ on a septic in P^3. But $x^7+y^7+z^7$ is semiquasihomogeneous (sqh), so we can also apply the stronger form of semicontinuity:

```
semicontsqh(s1,s2);
↦  17
```

So in fact a septic has at most 17 triple points of type $\tilde{E}_6$.

Note that `spectrumnd(f)` works only if f has a nondegenerate principal part. In fact `spectrumnd` will detect a degenerate principal part in many cases and print out an error message. However if it is known in advance that f has nondegenerate principal part, then the spectrum may be computed much faster using `spectrumnd(f,1)`.

C.6 Toric ideals and integer programming

C.6.1 Toric ideals

Let A denote an $m \times n$ matrix with integral coefficients. For $u \in \mathbb{Z}^n$, we define u^+, u^- to be the uniquely determined vectors with nonnegative coefficients and disjoint support (i.e., $u_i^+ = 0$ or $u_i^- = 0$ for each component i) such that $u = u^+ - u^-$. For $u \geq 0$ component-wise, let x^u denote the monomial $x_1^{u_1} \cdot \ldots \cdot x_n^{u_n} \in K[x_1, \ldots, x_n]$.

The ideal

$$I_A := (x^{u^+} - x^{u^-} | u \in \ker(A) \cap \mathbb{Z}^n) \ \subset K[x_1, \ldots, x_n]$$

is called a **toric ideal.**

The first problem in computing toric ideals is to find a finite generating set: Let $v_1, \ldots, v_r$ be a lattice basis of $\ker(A) \cap \mathbb{Z}^n$ (i.e, a basis of the $\mathbb{Z}$-module). Then

$$I_A := I : (x_1 \cdot \ldots \cdot x_n)^\infty$$

where

$$I =< x^{v_i^+} - x^{v_i^-} | i = 1, \ldots, r >$$

The required lattice basis can be computed using the LLL-algorithm (Section 5.1.155 [system], page 275, see see [[Coh93]], page 784). For the computation of the saturation, there are various possibilities described in the section Algorithms.

C.6.2 Algorithms

The following algorithms are implemented in Section D.4.35 [toric_lib], page 1387.

C.6.2.1 The algorithm of Conti and Traverso

The algorithm of Conti and Traverso (see [[CoTr91]], page 784) computes I_A via the extended matrix $B = (I_m | A)$, where I_m is the $m \times m$ unity matrix. A lattice basis of B is given by the set of vectors $(a^j, -e_j) \in \mathbb{Z}^{m+n}$, where a^j is the j-th row of A and e_j the j-th coordinate vector. We look at the ideal in $K[y_1, \ldots, y_m, x_1, \ldots, x_n]$ corresponding to these vectors, namely

$$I_1 =< y^{a_j^+} - x_j y^{a_j^-} | j = 1, \ldots, n > .$$

We introduce a further variable t and adjoin the binomial $t \cdot y_1 \cdot \ldots \cdot y_m - 1$ to the generating set of I_1, obtaining an ideal I_2 in the polynomial ring $K[t, y_1, \ldots, y_m, x_1, \ldots, x_n]$. I_2 is saturated w.r.t. all variables because all variables are invertible modulo I_2. Now I_A can be computed from I_2 by eliminating the variables $t, y_1, \ldots, y_m$.

Because of the big number of auxiliary variables needed to compute a toric ideal, this algorithm is rather slow in practice. However, it has a special importance in the application to integer programming (see Section C.6.4 [Integer programming], page 783).

C.6.2.2 The algorithm of Pottier

The algorithm of Pottier (see [[Pot94]], page 784) starts by computing a lattice basis $v_1, \ldots, v_r$ for the integer kernel of A using the LLL-algorithm (Section 5.1.155 [system], page 275). The ideal corresponding to the lattice basis vectors

$$I_1 =< x^{v_i^+} - x^{v_i^-} | i = 1, \ldots, r >$$

is saturated – as in the algorithm of Conti and Traverso – by inversion of all variables: One adds an auxiliary variable t and the generator $t \cdot x_1 \cdot \ldots \cdot x_n - 1$ to obtain an ideal I_2 in $K[t, x_1, \ldots, x_n]$ from which one computes I_A by elimination of t.

C.6.2.3 The algorithm of Hosten and Sturmfels

The algorithm of Hosten and Sturmfels (see [[HoSt95]], page 784) allows to compute I_A without any auxiliary variables, provided that A contains a vector w with positive coefficients in its row space. This is a real restriction, i.e., the algorithm will not necessarily work in the general case.

A lattice basis $v_1, \ldots, v_r$ is again computed via the LLL-algorithm. The saturation step is performed in the following way: First note that w induces a positive grading w.r.t. which the ideal

$$I =< x^{v_i^+} - x^{v_i^-} | i = 1, \ldots, r >$$

is homogeneous corresponding to our lattice basis. We use the following lemma:

Let I be a homogeneous ideal w.r.t. the weighted reverse lexicographical ordering with weight vector w and variable order $x_1 > x_2 > \ldots > x_n$. Let G denote a Groebner basis of I w.r.t. this ordering. Then a Groebner basis of $(I : x_n^\infty)$ is obtained by dividing each element of G by the highest possible power of x_n.

From this fact, we can succesively compute

$$I_A = I : (x_1 \cdot \ldots \cdot x_n)^\infty = (((I : x_1^\infty) : x_2^\infty) : \ldots : x_n^\infty);$$

in the i-th step we take x_i as the smallest variable and apply the lemma with x_i instead of x_n.

This procedure involves n Groebner basis computations. Actually, this number can be reduced to at most $n/2$ (see [[HoSh98]], page 784), and each computation – except for the first one – proves to be simple and fast in practice.

C.6.2.4 The algorithm of Di Biase and Urbanke

Like the algorithm of Hosten and Sturmfels, the algorithm of Di Biase and Urbanke (see [[DBUr95]], page 784) performs up to $n/2$ Groebner basis computations. It needs no auxiliary variables, but a supplementary precondition; namely, the existence of a vector without zero components in the kernel of A.

The main idea comes from the following observation:

Let B be an integer matrix, $u_1, \ldots, u_r$ a lattice basis of the integer kernel of B. Assume that all components of u_1 are positive. Then

$$I_B =< x^{u_i^+} - x^{u_i^-} | i = 1, \ldots, r >,$$

i.e., the ideal on the right is already saturated w.r.t. all variables.

The algorithm starts by finding a lattice basis $v_1, \ldots, v_r$ of the kernel of A such that v_1 has no zero component. Let $\{i_1, \ldots, i_l\}$ be the set of indices i with $v_{1,i} < 0$. Multiplying the components

$i_1, \ldots, i_l$ of $v_1, \ldots, v_r$ and the columns $i_1, \ldots, i_l$ of A by -1 yields a matrix B and a lattice basis $u_1, \ldots, u_r$ of the kernel of B that fulfill the assumption of the observation above. It is then possible to compute a generating set of I_A by applying the following "variable flip" successively to $i = i_1, \ldots, i_l$:

Let $>$ be an elimination ordering for x_i. Let A_i be the matrix obtained by multiplying the i-th column of A by -1. Let

$$\{x_i^{r_j} x^{a_j} - x^{b_j} | j \in J\}$$

be a Groebner basis of I_{A_i} w.r.t. $>$ (where x_i is neither involved in x^{a_j} nor in x^{b_j}). Then

$$\{x^{a_j} - x_i^{r_j} x^{b_j} | j \in J\}$$

is a generating set for I_A.

C.6.2.5 The algorithm of Bigatti, La Scala and Robbiano

The algorithm of Bigatti, La Scala and Robbiano (see [[BLR98]], page 784) combines the ideas of the algorithms of Pottier and of Hosten and Sturmfels. The computations are performed on a graded ideal with one auxiliary variable u and one supplementary generator $x_1 \cdot \ldots \cdot x_n - u$ (instead of the generator $t \cdot x_1 \cdot \ldots \cdot x_n - 1$ in the algorithm of Pottier). The algorithm uses a quite unusual technique to get rid of the variable u again.

There is another algorithm of the authors which tries to parallelize the computations (but which is not implemented in this library).

C.6.3 The Buchberger algorithm for toric ideals

Toric ideals have a very special structure that allows us to improve the Buchberger algorithm in many aspects: They are prime ideals and generated by binomials. Pottier used this fact to describe all operations of the Buchberger algorithm on the ideal generators in terms of vector additions and subtractions. Some other strategies like multiple reduction (see [[CoTr91]], page 784) or the use of bit vectors to represent the support of a monomial (see [[Big97]], page 784) may be applied to more general ideals, but show to be especially useful in the toric case.

C.6.4 Integer programming

Let A be an $m \times n$ matrix with integral coefficients, $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$. The problem

$$\min\{c^T x | x \in \mathbb{Z}^n, Ax = b, x \geq 0 \text{ component-wise}\}$$

is called an instance of the **integer programming problem** or **IP problem.**

The IP problem is very hard; namely, it is NP-complete.

For the following discussion let $c \geq 0$ (component-wise). We consider c as a weight vector; because of its nonnegativity, c can be refined into a monomial ordering $>_c$. It turns out that we can solve such an IP instance with the help of toric ideals:

First we assume that an initial solution v (i.e., $v \in \mathbb{Z}^n, v \geq 0, Av = b$) is already known. We obtain the optimal solution v_0 (i.e., with $c^T v_0$ minimal) by the following procedure:

- (1) Compute the toric ideal I(A) using one of the algorithms in the previous section.
- (2) Compute the reduced Groebner basis G(c) of I(A) w.r.t. $>_c$.
- (3) Reduce x^v modulo G(c) using the Hironaka division algorithm. If the result of this reduction is x^w , then w is an optimal solution of the given instance.

If no initial solution is known, we are nevertheless able to solve the problem with similar techniques. For this purpose we replace our instance by an extended instance with the matrix used in the Conti-Traverso algorithm. Indeed, the Conti-Traverso algorithm offers the possibility to verify solvability of a given instance and to find an initial solution in the case of existence (but none of the other algorithms does!). Details can be found in see [[CoTr91]], page 784 and see [[The99]], page 784.

An implementation of the above algorithm and some examples can be found in Section D.4.13 [intprog_lib], page 1118.

In general, classical methods for solving IP instances like Branch-and-Bound methods seem to be faster than the methods using toric ideals. But the latter have one great advantage: If one wants to solve various instances that differ only by the vector b , one has to perform steps (1) and (2) above only once. As the running time of step (3) is very short, solving all the instances is not much harder than solving one single instance.

For a detailed discussion see see [[The99]], page 784.

C.6.5 Relevant References

- [Big97] Bigatti, A.M.: Computation of Hilbert-Poincare series. Journal of Pure and Applied Algebra (1997) 199, 237-253
- [BLR98] Bigatti, A.M.; La Scala, R.; Robbiano, L.: Computing toric ideals. Journal of Symbolic Computation (1999) 27, 351-366
- [Coh93] Cohen, H.: A Course in Computational Algebraic Number Theory. Springer (1997)
- [CoTr91] Conti, P.; Traverso, C.: Buchberger algorithm and integer programming. Proceedings AAECC-9 (new Orleans), Springer LNCS (1991) 539, 130-139
- [DBUr95] Di Biase, F.; Urbanke, R.: An algorithm to calculate the kernel of certain polynomial ring homomorphisms. Experimental Mathematics (1995) 4, 227-234
- [HoSh98] Hosten, S.; Shapiro, J.: Primary decomposition of lattice basis ideals. Journal of Symbolic Computation (2000), 29, 625-639
- [HoSt95] Hosten, S.; Sturmfels, B.: GRIN: An implementation of Groebner bases for integer programming. in Balas, E.; Clausen, J. (editors): Integer Programming and Combinatorial Optimization. Springer LNCS (1995) 920, 267-276
- [Pot94] Pottier, L.: Groebner bases of toric ideals. Rapport de recherche 2224 (1997), INRIA Sophia Antipolis
- [Stu96] Sturmfels, B.: Groebner Bases and Convex Polytopes. University Lecture Series, Volume 8 (1996), American Mathematical Society
- [The99] Theis, C.: Der Buchberger-Algorithmus fuer torische Ideale und seine Anwendung in der ganzzahligen Optimierung. Diplomarbeit, Universitaet des Saarlandes (1999), Saarbruecken (Germany)

C.7 Non-commutative algebra

See Section 7.4 [Mathematical background (plural)], page 365, Section 7.9 [Mathematical background (letterplace)], page 635.

C.8 Decoding codes with Groebner bases

This section introduces some of the mathematical notions, definitions, and results for solving decoding problems and finding the minimum distance of linear (and in particular cyclic) codes. The material presented here should assist the user in working with Section D.10.2 [decodegb_lib], page 1937. More details can be obtained from [[BP2008b]], page 790.

C.8.1 Codes and the decoding problem

Codes

- Let F_q be a field with q elements. A *linear code* C is a linear subspace of F_q^n endowed with the **Hamming metric**.
- **Hamming distance** between **x**,**y**$\in F_q^n : d(x,y) = \#\{i | x_i \neq y_i\}$. **Hamming weight** of **x**$\in F_q^n$: $wt(x) = \#\{i | x_i \neq 0\}$.
- **Minimum distance** of the code $C : d(C) := \min_{\mathbf{x},\mathbf{y}\in C, \mathbf{x}\neq\mathbf{y}}(d(\mathbf{x},\mathbf{y}))$.
- The code C of dimension k and minimum distance d is denoted as $[n,k,d]$.
- A matrix G whose rows are the base vectors of C is the **generator matrix**.
- A matrix H with the property $\mathbf{c} \in C \iff H\mathbf{c}^T = 0$ is the **check matrix**.

Cyclic codes

The code C is **cyclic**, if for every codeword $\mathbf{c} = (c_0, \ldots, c_{n-1})$ in C its cyclic shift $(c_{n-1}, c_0, \ldots, c_{n-2})$ is again a codeword in C. When working with cyclic codes, vectors are usually presented as polynomials. So $\mathbf{c}$ is represented by the polynomial $c(x) = \sum_{i=0}^{n-1} c_i x^i$ with $x^n = 1$, more precisely $c(x)$ is an element of the factor ring $F_q[X]/\langle X^n - 1\rangle$. Cyclic codes over F_q of length n correspond one-to-one to ideals in this factor ring. We assume for cyclic codes that $(q,n) = 1$. Let $F = F_{q^m}$ be the splitting field of $X^n - 1$ over F_q. Then F has a **primitive n-th root of unity** which will be denoted by a. A cyclic code is uniquely given by a **defining set** S_C which is a subset of $\mathbb{Z}_n$ such that

$$c(x) \in C \;\text{ if }\; c(a^i) = 0 \;\text{ for all }\; i \in S_C.$$

A cyclic code has several defining sets.

Decoding problem

- **Complete decoding**: Given $y \in F_q^n$ and a code $C \subseteq F_q^n$, so that y is at distance $d(y,C)$ from the code, find $c \in C : d(y,c) = d(y,C)$.
- **Bounded up to half the minimum distance**: With the additional assumption $d(\mathbf{y},C) \leq (d(C)-1)/2$, a codeword with the above property is unique.

Decoding via systems solving

One distinguishes between two concepts:

- **Generic decoding**: Solve some system $S(C)$ and obtain some "closed" formulas CF. Evaluating these formulas at data specific to a received word $\mathbf{r}$ should yield a solution to the decoding problem. For example for $f \in CF : f(\mathit{syndrome}(\mathbf{r}), x) = \mathit{poly}(x)$. The roots of $\mathit{poly}(x) = 0$ yield error positions, see the section on the general error-locator polynomial.
- **Online decoding**: Solve some system $S(C, \mathbf{r})$. The solutions should solve the decoding problem.

Computational effort

- Generic decoding. Here, preprocessing is very hard, whereas decoding is relatively simple (if the formulas are sparse).
- Online decoding. In this case, decoding is the hard part.

C.8.2 Cooper philosophy

Computing syndromes in cyclic code case

Let C be an $[n,k]$ cyclic code over F_q; F is a splitting field with a being a primitive n-th root of unity. Let $S_C = \{i_1, \dots, i_{n-k}\}$ be the complete defining set of C. Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be a received word with $\mathbf{c} \in C$ and $\mathbf{e}$ an error vector. Denote the corresponding polynomials in $F_q[x]/\langle x^n - 1\rangle$ by $r(x)$, $c(x)$ and $e(x)$, resp. Compute syndromes

$$s_{i_m} = r(a^{i_m}) = e(a^{i_m}) = \sum_{l=1}^{t} e_{j_l}(a^{i_m})^{j_l}, \quad 1 \le m \le n-k,$$

where t is the number of errors, $j_1, \dots, j_t$ are the **error positions** and $e_{j_1}, \dots, e_{j_t}$ are the **error values**. Define $z_l = a^{j_l}$ and $y_l = e_{j_l}$. Then $z_1, \dots, z_t$ are the error locations and $y_1, \dots, y_t$ are the error values and the syndromes above become **generalized power sum functions** $s_{i_m} = \sum_{l=1}^{t} y_l z_l^{i_m}$, $1 \le m \le n-k$.

CRHT-ideal

Replace the concrete values above by variables and add some natural restrictions. Introduce

- $f_u := \sum_{l=1}^{e} Y_l Z_l^{i_u} - X_u = 0, 1 \le u \le n-k;$
- $\epsilon_j := X_j^{q^m} - X_j = 0, 1 \le j \le n-k$, since $s_j \in F$;
- $\eta_i := Z_i^{n+1} - Z_i = 0, 1 \le i \le e$, since a^{j_i} are either n-th roots of unity or zero;
- $\lambda_i := Y_i^{q-1} - 1 = 0, 1 \le i \le e$, since $y_l \in F_q \setminus \{0\}$.

We obtain the following set of polynomials in the variables $X = (X_1, \dots, X_{n-k})$, $Z = (Z_1, \dots, Z_e)$ and $Y = (Y_1, \dots, Y_e)$:

$$F_C = \{f_j, \epsilon_j, \eta_i, \lambda_i : 1 \le j \le n-k, 1 \le i \le e\} \subset F_q[X, Z, Y].$$

The zero-dimensional ideal I_C generated by F_C is the **CRHT-syndrome ideal** associated to the code C, and the variety $V(F_C)$ defined by F_C is the **CRHT-syndrome variety**, after Chen, Reed, Helleseth and Truong.

General error-locator polynomial

Adding some more polynomials to F_C, thus obtaining some F'_C, it is possible to prove the following **Theorem**:

Every cyclic code C possesses a **general error-locator polynomial** L_C from $F_q[X_1, \dots, X_{n-k}, Z]$ that satisfies the following two properties:

- $L_C = Z^e + a_{t-1}Z^{e-1} + \dots + a_0$ with $a_j \in F_q[X_1, \dots, X_{n-k}]$, $0 \le j \le e-1$, where e is the error-correcting capacity;
- given a syndrome $\mathbf{s} = (s_{i_1}, \dots, s_{i_{n-k}}) \in F^{n-k}$ corresponding to an error of weight $t \le e$ and error locations $\{k_1, \dots, k_t\}$, if we evaluate the $X_u = s_{i_u}$ for all $1 \le u \le n-k$, then the roots of $L_C(\mathbf{s}, Z)$ are exactly $a^{k_1}, \dots, a^{k_t}$ and 0 of multiplicity $e-t$, in other words $L_C(\mathbf{s}, Z) = Z^{e-t} \prod_{i=1}^{t} (Z - a^{k_i})$.

The general error-locator polynomial actually is an element of the reduced Gröbner basis of $\langle F'_C \rangle$. Having this polynomial, decoding of the cyclic code C reduces to univariate factorization.

For an example see `sysCRHT` in Section D.10.2 [decodegb_lib], page 1937. More on Cooper's philosophy and the general error-locator polynomial can be found in [[OS2005]], page 790.

Finding the minimum distance

The method described above can be adapted to find the minimum distance of a code. More concretely, the following holds:

Let C be the binary $[n, k, d]$ cyclic code with the defining set $S_C = \{i_1, \ldots, i_v\}$. Let $1 \leq w \leq n$ and let $J_C(w)$ denote the system:

$$Z_1^{i_1} + \ldots + Z_w^{i_1} = 0,$$

$$\vdots$$

$$Z_1^{i_v} + \ldots + Z_w^{i_v} = 0,$$

$$Z_1^n - 1 = 0,$$

$$\vdots$$

$$Z_w^n - 1 = 0,$$

$$p(n, Z_i, Z_j) = 0, 1 \leq i < j \leq w.$$

Then the number of solutions of $J_C(w)$ is equal to $w!$ times the number of codewords of weight w. And for $1 \leq w \leq d$, either $J_C(w)$ has no solutions, which is equivalent to $w < d$, or $J_C(w)$ has some solutions, which is equivalent to $w = d$.

For an example see `sysCRHTMindist` in Section D.10.2 [decodegb_lib], page 1937. More on finding the minimum distance with Groebner bases can be found in [[S2007]], page 790. See [[OS2005]], page 790, for the definition of the polynomial p above.

C.8.3 Generalized Newton identities

The **error-locator polynomial** is defined by

$$\sigma(Z) = \prod_{l=1}^{t} (Z - z_l).$$

If this product is expanded,

$$\sigma(Z) = Z^t + \sigma_1 Z^{t-1} + \ldots + \sigma_{t-1} Z + \sigma_t,$$

then the coefficients σ_i are the **elementary symmetric functions** in the error locations $z_1, \ldots, z_t$

$$\sigma_i = (-1)^i \sum_{1 \leq j_1 < j_2 < \ldots < j_i \leq t} z_{j_1} z_{j_2} \ldots z_{j_i}, \ 1 \leq i \leq t.$$

Generalized Newton identities

The syndromes $s_i = r(a^i) = e(a^i)$ and the coefficients σ_i satisfy the following **generalized Newton identities**:

$$s_i + \sum_{j=1}^{t} \sigma_j s_{i-j} = 0, \quad \text{for all} \ \ i \in \mathbb{Z}_n.$$

Decoding up to error-correcting capacity

We have $s_{i+n} = s_i$, for all $i \in \mathbb{Z}_n$, since $s_{i+n} = r(a^{i+n}) = r(a^i)$. Furthermore

$$s_i^q = (e(a^i))^q = e(a^{iq}) = s_{qi}, \text{ for all } i \in Z_n,$$

and $\sigma_i^{q^m} = \sigma_i$, for all $1 \leq i \leq t$. Replace the syndromes by variables and obtain the following set of polynomials $Newton_t$ in the variables $S_1, \ldots, S_n$ and $\sigma_1, \ldots, \sigma_t$:

$$\sigma_i^{q^m} - \sigma_i, \quad \forall\ 1 \leq i \leq t,$$

$$S_{i+n} - S_i, \quad \forall\ i \in \mathbb{Z}_n,$$

$$S_i^q - S_{qi}, \quad \forall\ i \in \mathbb{Z}_n,$$

$$S_i + \sum_{j=1}^{t} \sigma_j S_{i-j}, \quad \forall\ i \in \mathbb{Z}_n,$$

$$S_i - s_i(r) \quad \forall\ i \in S_C.$$

For an example see `sysNewton` in Section D.10.2 [decodegb_lib], page 1937. More on this method and the method based on Waring function can be found in [[ABF2002]], page 790. See also [[ABF2008]], page 790.

C.8.4 Fitzgerald-Lax method

Affine codes

Let $I = \langle g_1, \ldots, g_m \rangle \subseteq F_q[X_1, \ldots, X_s]$ be an ideal. Define

$$I_q := I + \langle X_1^q - X_1, \ldots, X_s^q - X_s \rangle.$$

So I_q is a zero-dimensional ideal. Define also $V(I_q) =: \{P_1, \ldots, P_n\}$. Every q-ary linear code C with parameters $[n, k]$ can be seen as an **affine variety code** $C(I, L)$, that is, the image of a vector space L of the **evaluation map**

$$\phi : R \to F_q^n$$

$$\bar{f} \mapsto (f(P_1), \ldots, f(P_n)),$$

where $R := F_q[U_1, \ldots, U_s]/I_q$, L is a vector subspace of R and $\bar{f}$ the coset of f in $F_q[U_1, \ldots, U_s]$ modulo I_q.

Decoding affine variety codes

Given a q-ary $[n, k]$ code C with a generator matrix $G = (g_{ij})$:

1. choose s, such that $q^s \geq n$, and construct s distinct points $P_1, \ldots, P_s$ in F_q^s.
2. Construct a Gröbner basis $\{g_1, \ldots, g_m\}$ for an ideal I of polynomials from $F_q[X_1, \ldots, X_s]$ that vanish at the points $P_1, \ldots, P_s$. Define $\xi_i \in F_q[X_1, \ldots, X_s]$ such that $\xi_i(P_i) = 1, \xi_i(P_j) = 0, i \neq j$.
3. Then $f_i = \sum_{i=1}^{n} g_{ij} \xi_j$ span the space L, so that $g_{ij} = f_i(P_j)$.

In this way we obtain that the code C is the image of the evaluation above, thus $C = C(I, L)$. In the same way by considering a parity check matrix instead of a generator matrix we have that the dual code is also an affine variety code.

The method of decoding is a generalization of CRHT. One needs to add polynomials $(g_l(X_{k1}, \ldots, X_{ks}))_{l=1,\ldots,m;k=1,\ldots,t}$ for every error position. We also assume that field equations on X_{ij}'s are included among the polynomials above. Let C be a q-ary $[n, k]$ linear code such that its dual is written as an affine variety code of the form $C^\perp = C(I, L)$. Let $\mathbf{r} = \mathbf{c}+\mathbf{e}$ as usual and $t \le e$. Then the syndromes are computed by $s_i = \sum_{j=1}^n r_j f_i(P_j) = \sum_{j=1}^n e_j f_i(P_j)$ for $i = 1, \ldots, n-k$.

Consider the ring $F_q[X_{11}, \ldots, X_{1s}, \ldots, X_{t1}, \ldots, X_{ts}, E_1, \ldots, E_t]$, where $(X_{i1}, \ldots, X_{is})$ correspond to the i-th error position and E_i to the i-th error value. Consider the ideal Id_C generated by

$$\sum_{j=1}^{t} E_j f_i(X_{j1}, \ldots, X_{js}) - s_i, 1 \le i \le n-k,$$

$$g_l(X_{j1}, \ldots, X_{js}), 1 \le l \le m,$$

$$E_k^{q-1} - 1.$$

Theorem: Let G be the reduced Gröbner basis for Id_C with respect to an elimination order $X_{11} < \ldots < X_{1s} < E_1$. Then we may solve for the error locations and values by applying elimination theory to the polynomials in G.

For an example see `sysFL` in Section D.10.2 [decodegb_lib], page 1937. More on this method can be found in [[FL1998]], page 790.

C.8.5 Decoding method based on quadratic equations

Preliminary definitions

Let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ be a basis of F_q^n and let B be the $n \times n$ matrix with $\mathbf{b}_1, \ldots, \mathbf{b}_n$ as rows. The **unknown syndrome** $\mathbf{u}(B, \mathbf{e})$ of a word $\mathbf{e}$ w.r.t B is the column vector $\mathbf{u}(B, \mathbf{e}) = B\mathbf{e}^T$ with entries $u_i(B, \mathbf{e}) = \mathbf{b}_i \cdot \mathbf{e}$ for $i = 1, \ldots, n$.

For two vectors $\mathbf{x}, \mathbf{y} \in F_q^n$ define $\mathbf{x} * \mathbf{y} = (x_1y_1, \ldots, x_ny_n)$. Then $\mathbf{b}_i * \mathbf{b_j}$ is a linear combination of $\mathbf{b}_1, \ldots, \mathbf{b}_n$, so there are constants $\mu_l^{ij} \in F_q$ such that $\mathbf{b}_i * \mathbf{b}_j = \sum_{l=1}^n \mu_l^{ij} \mathbf{b}_l$. The elements $\mu_l^{ij} \in F_q$ are the **structure constants** of the basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$.

Let B_s be the $s \times n$ matrix with $\mathbf{b}_1, \ldots, \mathbf{b}_s$ as rows ($B = B_n$). Then $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is an **ordered MDS basis** and B an **MDS matrix** if all the $s \times s$ submatrices of B_s have rank s for all $s = 1, \ldots, n$.

Expressing known syndromes

Let C be an F_q-linear code with parameters $[n, k, d]$. W.l.o.g $n \le q$. H is a check matrix of C. Let $\mathbf{h}_1, \ldots, \mathbf{h}_{n-k}$ be the rows of H. One can express $\mathbf{h}_i = \sum_{j=1}^n a_{ij} \mathbf{b}_j$ with some $a_{ij} \in F_q$. In other words $H = AB$ where A is the $(n-k) \times n$ matrix with entries a_{ij}.

Let $\mathbf{r} = \mathbf{c}+\mathbf{e}$ be a received word with $\mathbf{c} \in C$ and $\mathbf{e}$ an error vector. The syndromes of $\mathbf{r}$ and $\mathbf{e}$ w.r.t H are equal and known:

$$s_i(\mathbf{r}) := \mathbf{h}_i \cdot \mathbf{r} = \mathbf{h}_i \cdot \mathbf{e} = s_i(\mathbf{e}).$$

They can be expressed in the unknown syndromes of $\mathbf{e}$ w.r.t B:

$$s_i(\mathbf{r}) = s_i(\mathbf{e}) = \sum_{j=1}^{n} a_{ij} u_j(\mathbf{e})$$

since $\mathbf{h}_i = \sum_{j=1}^n a_{ij} \mathbf{b}_j$ and $\mathbf{b}_j \cdot \mathbf{e} = u_j(\mathbf{e})$.

Constructing the system

Let B be an MDS matrix with structure constants μ_l^{ij}. Define U_{ij} in the variables $U_1, \ldots, U_n$ by

$$U_{ij} = \sum_{l=1}^{n} \mu_l^{ij} U_l.$$

The ideal $J(\mathbf{r})$ in $F_q[U_1, \ldots, U_n]$ is generated by

$$\sum_{l=1}^{n} a_{jl} U_l - s_j(\mathbf{r}) \text{ for } j = 1, \ldots, n-k.$$

The ideal $I(t, U, V)$ in $F_q[U_1, \ldots, U_n, V_1, \ldots, V_t]$ is generated by

$$\sum_{j=1}^{t} U_{ij} V_j - U_{i,t+1} \text{ for } i = 1, \ldots, n$$

Let $J(t, \mathbf{r})$ be the ideal in $F_q[U_1, \ldots, U_n, V_1, \ldots, V_t]$ generated by $J(\mathbf{r})$ and $I(t, U, V)$.

Main theorem

Let B be an MDS matrix with structure constants μ_l^{ij}. Let H be a check matrix of the code C such that $H = AB$ as above. Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be a received word with $\mathbf{c} \in C$ the codeword sent and $\mathbf{e}$ the error vector. Suppose that $wt(\mathbf{e}) \neq 0$ and $wt(\mathbf{e}) \leq \lfloor (d(C) - 1)/2 \rfloor$. Let t be the smallest positive integer such that $J(t, \mathbf{r})$ has a solution $(\mathbf{u}, \mathbf{v})$ over the algebraic closure of F_q. Then

- $wt(\mathbf{e}) = t$ and the solution is unique and of multiplicity one satisfying $\mathbf{u} = \mathbf{u}(\mathbf{e})$.
- the reduced Gröbner basis G for the ideal $J(t, \mathbf{r})$ w.r.t any monomial ordering is

  $$U_i - u_i(\mathbf{e}), i = 1, \ldots, n,$$
  $$V_j - v_j, j = 1, \ldots, t,$$

  where $(\mathbf{u}(\mathbf{e}), \mathbf{v})$ is the unique solution.

For an example see `sysQE` in Section D.10.2 [decodegb_lib], page 1937. More on this method can be found in [[BP2008a]], page 790.

C.8.6 References for decoding with Groebner bases

- [ABF2002] Augot D.; Bardet M.; Faugére J.-C.: Efficient Decoding of (binary) Cyclic Codes beyond the correction capacity of the code using Gröbner bases. INRIA Report (2002) 4652
- [ABF2008] Augot D.; Bardet M.; Faugére, J.-C.: On the decoding of cyclic codes with Newton identities. to appear in Special Issue "Gröbner Bases Techniques in Cryptography and Coding Theory" of Journ. Symbolic Comp. (2008)
- [BP2008a] Bulygin S.; Pellikaan R.: Bounded distance decoding of linear error-correcting codes with Gröbner bases. to appear in Special Issue "Gröbner Bases Techniques in Cryptography and Coding Theory" of Journ. Symbolic Comp. (2008)
- [BP2008b] Bulygin S.; Pellikaan R.: Decoding and finding the minimum distance with Gröbner bases: history and new insights. to appear in "Selected topics of information and coding theory", World Scientific (2008)
- [FL1998] Fitzgerald J.; Lax R.F.: Decoding affine variety codes using Gröbner bases. Designs, Codes and Cryptography (1998) 13, 147-158
- [OS2005] Orsini E.; Sala M.: Correcting errors and erasures via the syndrome variety. J. Pure and Appl. Algebra (2005) 200, 191-226
- [S2007] Sala M.: Gröbner basis techniques to compute weight distributions of shortened cyclic codes. J. Algebra Appl. (2007) 6, 3, 403-414

C.9 References

The Centre for Computer Algebra Kaiserslautern publishes a series of preprints which are electronically available at `https://www.singular.uni-kl.de/reports`. Other sources to check are `http://symbolicnet.org/`, `http://www-sop.inria.fr/galaad/`,... and the following list of books.

For references on non-commutative algebras and algorithms, see Section 7.4.4 [References (plural)], page 369.

Text books on computational algebraic geometry

- Adams, W.; Loustaunau, P.: An Introduction to Gröbner Bases. Providence, RI, AMS, 1996
- Becker, T.; Weisspfenning, V.: Gröbner Bases - A Computational Approach to Commutative Algebra. Springer, 1993
- Cohen, H.: A Course in Computational Algebraic Number Theory, Springer, 1995
- Cox, D.; Little, J.; O'Shea, D.: Ideals, Varieties and Algorithms. Springer, 1996
- Cox, D.; Little, J.; O'Shea, D.: Using Algebraic Geometry. Springer, 1998
- Eisenbud, D.: Commutative Algebra with a View Toward Algebraic Geometry. Springer, 1995
- Greuel, G.-M.; Pfister, G.: A Singular Introduction to Commutative Algebra. Springer, 2002
- Mishra, B.: Algorithmic Algebra, Texts and Monographs in Computer Science. Springer, 1993
- Sturmfels, B.: Algorithms in Invariant Theory. Springer 1993
- Vasconcelos, W.: Computational Methods in Commutative Algebra and Algebraic Geometry. Springer, 1998

Descriptions of algorithms

- Bareiss, E.: Sylvester's identity and multistep integer-preserving Gaussian elimination. Math. Comp. 22 (1968), 565-578
- Campillo, A.: Algebroid curves in positive characteristic. SLN 813, 1980
- Chou, S.: Mechanical Geometry Theorem Proving. D.Reidel Publishing Company, 1988
- Decker, W.; Greuel, G.-M.; Pfister, G.: Primary decomposition: algorithms and comparisons. Preprint, Univ. Kaiserslautern, 1998. To appear in: Greuel, G.-M.; Matzat, B. H.; Hiss, G. (Eds.), Algorithmic Algebra and Number Theory. Springer Verlag, Heidelberg, 1998
- Decker, W.; Greuel, G.-M.; de Jong, T.; Pfister, G.: The normalisation: a new algorithm, implementation and comparisons. Preprint, Univ. Kaiserslautern, 1998
- Decker, W.; Heydtmann, A.; Schreyer, F. O.: Generating a Noetherian Normalization of the Invariant Ring of a Finite Group, 1997, to appear in Journal of Symbolic Computation
- Faugère, J. C.; Gianni, P.; Lazard, D.; Mora, T.: Efficient computation of zero-dimensional Gröbner bases by change of ordering. Journal of Symbolic Computation, 1989
- Gräbe, H.-G.: On factorized Gröbner bases, Univ. Leipzig, Inst. für Informatik, 1994
- Grassmann, H.; Greuel, G.-M.; Martin, B.; Neumann, W.; Pfister, G.; Pohl, W.; Schönemann, H.; Siebert, T.: On an implementation of standard bases and syzygies in Singular. Proceedings of the Workshop Computational Methods in Lie theory in AAECC (1995)
- Greuel, G.-M.; Pfister, G.: Advances and improvements in the theory of standard bases and syzygies. Arch. d. Math. 63(1995)
- Kemper; Generating Invariant Rings of Finite Groups over Arbitrary Fields. 1996, to appear in Journal of Symbolic Computation

- Kemper and Steel: Some Algorithms in Invariant Theory of Finite Groups. 1997
- Lee, H.R.; Saunders, B.D.: Fraction Free Gaussian Elimination for Sparse Matrices. Journal of Symbolic Computation (1995) 19, 393-402
- Schönemann, H.: Algorithms in Singular, Reports on Computer Algebra 2(1996), Kaiserslautern
- Siebert, T.: On strategies and implementations for computations of free resolutions. Reports on Computer Algebra 8(1996), Kaiserslautern
- Wang, D.: Characteristic Sets and Zero Structure of Polynomial Sets. Lecture Notes, RISC Linz, 1989

Appendix D SINGULAR libraries

Singular comes with a set of standard libraries. Their content is described in the following subsections.

Use the `LIB` command (see Section 5.1.79 [LIB], page 211) for loading of single libraries, and the command `LIB "all.lib";` for loading all libraries.

Interpreter libraries:

See also Section 7.5 [PLURAL libraries], page 370 and Section 7.10 [LETTERPLACE libraries], page 639.

D.1 standard_lib

The library `standard.lib` provides extensions to the set of built-in commands and is automatically loaded during the start of Singular, unless Singular is started up with the `--no-stdlib` command line option (see Section 3.1.6 [Command line options], page 19).

Library: standard.lib

Purpose: Procedures which are always loaded at Start-up

Procedures:

D.1.1 qslimgb

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Usage: qslimgb(i); i ideal or module

Return: same type as input, a standard basis of i computed with slimgb

Note: Only as long as slimgb does not know qrings qslimgb should be used in case the basering is (possibly) a quotient ring.
The quotient ideal is added to the input and slimgb is applied.

Example:

```
   ring R  = (0,v),(x,y,z,u),dp;
qring Q = std(x2-y3);
ideal i = x+y2,xy+yz+zu+u*v,xyzu*v-1;
ideal j = qslimgb(i); j;
↦ j[1]=y-1
↦ j[2]=x+1
↦ j[3]=(v)*z+(v2)*u+(-v-1)
↦ j[4]=(-v2)*u2+(v+1)*u+1
module m = [x+y2,1,0], [1,1,x2+y2+xyz];
print(qslimgb(m));
↦ y2+x,x2+xy,1,        0,    0,    -x,          -xy-xz-x,
↦ 1,   y,    1,        y3-x2,0,    y2-1,        y2z-xy-x-z,
↦ 0,   0,    xyz+x2+y2,0,    y3-x2,x2y2+x3z+x2y,x3z2-x3y-2*x3-xy2
```

D.1.2 par2varRing

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 793).

Usage: par2varRing([l]); l list of ideals/modules [default:l=empty list]

Return: list, say L, with L[1] a ring where the parameters of the basering have been converted to an additional last block of variables, all of weight 1, and ordering dp.
If a list l with l[i] an ideal/module is given, then
l[i] + minpoly*freemodule(nrows(l[i])) is mapped to an ideal/module in L[1] with name Id[i].
If the basering has no parameters then L[1] is the basering.

Example:

```
   ring R = (0,x),(y,z,u,v),lp;
minpoly = x2+1;
ideal i = x3,x2+y+z+u+v,xyzuv-1; i;
↦ i[1]=(-x)
↦ i[2]=y+z+u+v-1
↦ i[3]=(x)*yzuv-1
def P = par2varRing(i)[1]; P;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering lp
↦ //                  : names    y z u v
↦ //        block   2 : ordering dp
↦ //                  : names    x
↦ //        block   3 : ordering C
setring(P);
Id[1];
↦ _[1]=-x
↦ _[2]=y+z+u+v-1
↦ _[3]=yzuvx-1
↦ _[4]=x2+1
setring R;
module m = x3*[1,1,1], (xyzuv-1)*[1,0,1];
def Q = par2varRing(m)[1]; Q;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering lp
↦ //                  : names    y z u v
↦ //        block   2 : ordering dp
↦ //                  : names    x
↦ //        block   3 : ordering C
setring(Q);
print(Id[1]);
↦ -x,yzuvx-1,x2+1,0,   0,
↦ -x,0,      0,   x2+1,0,
↦ -x,yzuvx-1,0,   0,   x2+1
```

D.2 General purpose

D.2.1 all_lib

The library `all.lib` provides a convenient way to load all libraries of the Singular distribution.

Example:

```
option(loadLib);
LIB "all.lib";
```

```
↦ // ** loaded all.lib (4.1.1.0,Jan_2018)
↦ // ** loaded ratgb.lib (4.1.2.0,Feb_2019)
↦ // ** loaded qmatrix.lib (4.1.2.0,Feb_2019)
↦ // ** loaded purityfiltration.lib (4.1.2.0,Feb_2019)
↦ // ** loaded perron.lib (4.1.2.0,Feb_2019)
↦ // ** loaded nctools.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ncpreim.lib (4.1.2.0,Feb_2019)
↦ // ** loaded dmodloc.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ncModslimgb.lib (4.1.3.0,Apr_2020)
↦ // ** loaded resources.lib (4.1.2.0,Feb_2019)
↦ // ** loaded parallel.lib (4.1.2.0,Feb_2019)
↦ // ** loaded tasks.lib (4.1.2.0,Feb_2019)
↦ // ** loaded nchomolog.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ncfactor.lib (4.3.1.3,Jan_2023)
↦ // ** loaded ncdecomp.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ncalg.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ncall.lib (4.1.2.0,Feb_2019)
↦ // ** loaded involut.lib (4.1.2.0,Feb_2019)
↦ // ** loaded gkdim.lib (4.1.2.0,Feb_2019)
↦ // ** loaded freegb.lib (4.1.2.0,Feb_2019)
↦ // ** loaded Singular/.libs/../MOD/freealgebra.so
↦ // ** loaded fpaprops.lib (4.1.2.0,Feb_2019)
↦ // ** loaded fpalgebras.lib (4.1.2.0,Feb_2019)
↦ // ** loaded fpadim.lib (4.1.2.0,Feb_2019)
↦ // ** loaded dmodideal.lib (4.1.2.0,Feb_2019)
↦ // ** loaded dmodvar.lib (4.1.2.0,Feb_2019)
↦ // ** loaded dmodapp.lib (4.1.3.0,Mar_2020)
↦ // ** loaded dmod.lib (4.1.2.0,Feb_2019)
↦ // ** loaded central.lib (4.1.2.0,Feb_2019)
↦ // ** loaded bfun.lib (4.3.1.3,Feb_2023)
↦ // ** loaded bimodules.lib (4.1.2.0,Feb_2019)
↦ // ** loaded zeroset.lib (4.3.2.3,Jun_2023)
↦ // ** loaded weierstr.lib (4.1.2.0,Feb_2019)
↦ // ** loaded triang.lib (4.1.2.0,Feb_2019)
↦ // ** loaded toric.lib (4.3.1.3,Feb_2023)
↦ // ** loaded teachstd.lib (4.1.2.0,Feb_2019)
↦ // ** loaded surfex.lib (4.1.2.0,Feb_2019)
↦ // ** loaded surfacesignature.lib (4.1.2.0,Feb_2019)
↦ // ** loaded surf_jupyter.lib (4.1.2.0,Feb_2019)
↦ // ** loaded surf.lib (4.3.1.2,Sep_2022)
↦ // ** loaded stratify.lib (4.1.2.0,Feb_2019)
↦ // ** loaded stanleyreisner.lib (4.2.0.0,Dec_2020)
↦ // ** loaded Singular/.libs/../MOD/cohomo.so
↦ // ** loaded spectrum.lib (4.1.2.0,Feb_2019)
↦ // ** loaded spcurve.lib (4.3.1.3,Jan_2023)
↦ // ** loaded solve.lib (4.3.1.3,Feb_2023)
↦ // ** loaded signcond.lib (4.1.2.0,Feb_2019)
↦ // ** loaded sing4ti2.lib (4.2.1,Nov_2021)
↦ // ** loaded sing.lib (4.2.0.2,May_2021)
↦ // ** loaded sheafcoh.lib (4.3.2.9,Oct_2023)
↦ // ** loaded sagbi.lib (4.3.1.2,Nov_2022)
↦ // ** loaded rootsur.lib (4.1.2.0,Feb_2019)
↦ // ** loaded rootsmr.lib (4.1.2.0,Feb_2019)
↦ // ** loaded rinvar.lib (4.3.1.3,Feb_2023)
↦ // ** loaded ringgb.lib (4.2.0.0,Dec_2020)
↦ // ** loaded ring.lib (4.3.1.3,Feb_2023)
↦ // ** loaded reszeta.lib (4.3.1.3,Feb_2023)
↦ // ** loaded resolve.lib (4.3.1.3,Feb_2023)
↦ // ** loaded resjung.lib (4.1.2.0,Feb_2019)
↦ // ** loaded resgraph.lib (4.1.2.0,Feb_2019)
↦ // ** loaded resbinomial.lib (4.1.2.0,Feb_2019)
↦ // ** loaded reesclos.lib (4.1.2.0,Feb_2019)
↦ // ** loaded redcgs.lib (4.2.0.0,Dec_2020)
↦ // ** loaded realrad.lib (4.1.2.0,Feb_2019)
↦ // ** loaded realclassify.lib (4.2.1.0,Jul_2021)
↦ // ** loaded rootisolation.lib (4.2.1.0,Jul_2021)
↦ // ** loaded Singular/.libs/../MOD/interval.so
↦ // ** loaded classify2.lib (4.1.2.0,Feb_2019)
↦ // ** loaded gfa.lib.so
↦ // ** loaded polyclass.lib (4.2.0.0,Dec_2020)
↦ // ** loaded random.lib (4.1.2.0,Feb_2019)
↦ // ** loaded qhmoduli.lib (4.1.2.0,Feb_2019)
↦ // ** loaded primitiv.lib (4.1.2.0,Feb_2019)
↦ // ** loaded primdecint.lib (4.3.2.2,Nov_2022)
↦ // ** loaded primdec.lib (4.3.2.9,Oct_2023)
↦ // ** loaded presolve.lib (4.3.1.3,Feb_2023)
↦ // ** loaded polylib.lib (4.2.0.0,Dec_2020)
↦ // ** loaded pointid.lib (4.4.0.0,Nov_2023)
↦ // ** loaded phindex.lib (4.1.2.0,Feb_2019)
↦ // ** loaded pfd.lib (4.1.3.2,Aug_2020)
↦ // ** loaded paraplanecurves.lib (4.3.2.10,Nov_2023)
↦ // ** loaded ntsolve.lib (4.1.2.0,Feb_2019)
↦ // ** loaded normaliz.lib (4.3.1.0,June_2022)
↦ // ** loaded normal.lib (4.3.2.10,Nov_2023)
↦ // ** loaded curveInv.lib (4.1.2.0,Feb_2019)
↦ // ** loaded noether.lib (4.1.2.0,Feb_2019)
↦ // ** loaded nfmodsyz.lib (4.1.2.0,Feb_2019)
↦ // ** loaded nfmodstd.lib (4.1.2.0,Feb_2019)
↦ // ** loaded mregular.lib (4.1.2.0,Feb_2019)
↦ // ** loaded mprimdec.lib (4.1.2.0,Feb_2019)
↦ // ** loaded monomialideal.lib (4.1.2.0,Feb_2019)
↦ // ** loaded mondromy.lib (4.1.2.0,Feb_2019)
↦ // ** loaded modstd.lib (4.3.0.1,Mar_2022)
↦ // ** loaded modular.lib (4.2.0.0,Dec_2020)
↦ // ** loaded moddiq.lib (4.3.2.2,Jun_2023)
↦ // ** loaded matrix.lib (4.3.2.3,Aug_2023)
↦ // ** loaded makedbm.lib (4.1.2.0,Feb_2019)
↦ // ** loaded linalg.lib (4.3.1.3,Feb_2023)
↦ // ** loaded latex.lib (4.1.2.0,Feb_2019)
↦ // ** loaded kskernel.lib (4.1.2.0,Feb_2019)
↦ // ** loaded jacobson.lib (4.1.2.0,Feb_2019)
↦ // ** loaded intprog.lib (4.2.1.3,Dec_2021)
↦ // ** loaded inout.lib (4.1.2.0,Feb_2019)
↦ // ** loaded integralbasis.lib (4.3.2.10,Nov_2023)
↦ // ** loaded puiseuxexpansions.lib (4.3.1.0,Jul_2022)
↦ // ** loaded hyperel.lib (4.1.2.0,Feb_2019)
↦ // ** loaded homolog.lib (4.3.1.3,Feb_2023)
↦ // ** loaded hnoether.lib (4.1.2.0,Feb_2019)
↦ // ** loaded grwalk.lib (4.1.2.0,Feb_2019)
↦ // ** loaded groups.lib (4.1.2.0,Feb_2019)
↦ // ** loaded grobcov.lib (4.2.0,February_2021)
↦ // ** loaded graphics.lib (4.1.2.0,Feb_2019)
↦ // ** loaded gmssing.lib (4.1.2.0,Feb_2019)
↦ // ** loaded gmspoly.lib (4.1.2.0,Feb_2019)
↦ // ** loaded general.lib (4.1.2.0,Feb_2019)
↦ // ** loaded finvar.lib (4.3.1.3,Feb_2023)
↦ // ** loaded equising.lib (4.1.2.0,Feb_2019)
↦ // ** loaded elim.lib (4.3.2.5,Jul_2023)
↦ // ** loaded deform.lib (4.3.1.3,Jan_2023)
↦ // ** loaded decodegb.lib (4.3.1.3,Feb_2023)
↦ // ** loaded curvepar.lib (4.1.2.0,Feb_2019)
↦ // ** loaded crypto.lib (4.2.1.0,Jul_2021)
↦ // ** loaded control.lib (4.1.2.0,Feb_2019)
↦ // ** loaded compregb.lib (4.1.2.0,Feb_2019)
↦ // ** loaded classify.lib (4.1.2.0,Feb_2019)
↦ // ** loaded cisimplicial.lib (4.1.2.0,Feb_2019)
↦ // ** loaded brnoeth.lib (4.3.1.3,Jan_2023)
↦ // ** loaded assprimeszerodim.lib (4.1.2.0,Feb_2019)
↦ // ** loaded arcpoint.lib (4.3.1.3,Jan_2023)
↦ // ** loaded algebra.lib (4.3.1.3,Feb_2023)
↦ // ** loaded alexpoly.lib (4.1.2.0,Feb_2019)
↦ // ** loaded aksaka.lib (4.1.2.0,Feb_2019)
↦ // ** loaded ainvar.lib (4.3.1.3,Feb_2023)
↦ // ** loaded absfact.lib (4.1.2.0,Feb_2019)
```

D.2.2 compregb_lib

Library: compregb.lib

Purpose: experimental implementation for comprehensive Groebner systems

Author: Akira Suzuki (http://kurt.scitec.kobe-u.ac.jp/~sakira/CGBusingGB/) (<sakira@kobe-u.ac.jp>)

Overview: see "A Simple Algorithm to compute Comprehensive Groebner Bases using Groebner Bases" by Akira Suzuki and Yosuke Sato for details.

Procedures:

D.2.2.1 cgs

Procedure from library `compregb.lib` (see Section D.2.2 [compregb_lib], page 797).

Usage: cgs(Polys,Vars,Paras,RingVar,RingAll); Polys an ideal, Vars, the list of variables, Paras the list of parameters, RingVar the ring with Paras as parameters, RingAll the ring with Paras as variables (RingAll should be the current ring)

Return: a list L of lists L[i] of a polynomial and an ideal:
L[i][1] the polynomial giving the condition on the parameters L[i][2] the Groebner basis for this case

Example:

```
LIB "compregb.lib";
ring RingVar=(0,a,b),(x,y,t),lp;
ring RingAll=0,(x,y,t,a,b),(lp(3),dp);
ideal polys=x^3-a,y^4-b,x+y-t;
list vars=x,y,t;
list paras=a,b;
list G = cgs(polys,vars,paras,RingVar,RingAll);
G;
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       _[1]=b
↦       _[2]=a
↦       _[3]=t6
↦       _[4]=5yt4-3t5
↦       _[5]=6y2t2-8yt3+3t4
↦       _[6]=y3-3y2t+3yt2-t3
↦       _[7]=x+y-t
↦ [2]:
↦    [1]:
↦       a
↦    [2]:
↦       _[1]=b
↦       _[2]=a4
↦       _[3]=t6a3
↦       _[4]=5t8a2-28t5a3
↦       _[5]=14t10a-60t7a2+105t4a3
↦       _[6]=t12-4t9a+6t6a2-4t3a3
↦       _[7]=81ya3-14t10+60t7a-105t4a2+59ta3
↦       _[8]=81yt2a2+4t9-21t6a+3t3a2+14a3
↦       _[9]=21yt3a+6ya2-t7-7t4a+8ta2
↦       _[10]=12yt5+15yt2a-7t6+5t3a+2a2
↦       _[11]=3y2a+5yt4+4yta-3t5+3t2a
↦       _[12]=6y2t2-8yt3-ya+3t4-3ta
↦       _[13]=y3-3y2t+3yt2-t3+a
↦       _[14]=x+y-t
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       _[1]=b
↦       _[2]=a
↦       _[3]=t6
↦       _[4]=5yt4-3t5
↦       _[5]=6y2t2-8yt3+3t4
↦       _[6]=y3-3y2t+3yt2-t3
↦       _[7]=x+y-t
↦ [4]:
↦    [1]:
↦       b
↦    [2]:
↦       _[1]=a
↦       _[2]=b3
↦       _[3]=t6b2
↦       _[4]=5t9b-18t5b2
↦       _[5]=t12-3t8b+3t4b2
↦       _[6]=32yb2-5t9+18t5b-45tb2
↦       _[7]=32yt3b+3t8-30t4b-5b2
↦       _[8]=5yt4+3yb-3t5-5tb
↦       _[9]=10y2b-24ytb-t6+15t2b
↦       _[10]=6y2t2-8yt3+3t4-b
↦       _[11]=y3-3y2t+3yt2-t3
↦       _[12]=x+y-t
↦ [5]:
↦    [1]:
↦       ab
↦    [2]:
↦       _[1]=729a4-4096b3
↦       _[2]=41472t11b2-6561t10a3+5832t9a2b-171072t8ab2+27648t7b3-4374t6a3b\
  +252720t5a2b2-2215296t4ab3+2093568t3b4-497097t2a3b2-802296ta2b3+215488ab4
↦       _[3]=46656t11ab-41472t10b2+6561t9a3-192456t8a2b+31104t7ab2-27648t6b\
  3+284310t5a3b-2492208t4a2b2+2355264t3ab3-3142144t2b4-902583ta3b2+242424a2\
  b3
↦       _[4]=52488t11a2-46656t10ab+41472t9b2-216513t8a3+34992t7a2b-31104t6a\
  b2+1797120t5b3-2803734t4a3b+2649672t3a2b2-3534912t2ab3-5705216tb4+272727a\
  3b2
↦       _[5]=729t12-2916t9a-2187t8b+4374t6a2-34992t5ab+2187t4b2-2916t3a3-21\
  870t2a2b-8748tab2+3367b3
↦       _[6]=3594240ytb3+568620ya3b-99144t11a-1728t10b+426465t8a2+327888t7a\
  b+17280t6b2-752328t5a3+4509270t4a2b-366984t3ab2+2206528t2b3+1791180ta3b+6\
  59529a2b2
↦       _[7]=1137240yta2b2+1010880yab3-28431t10a2+24786t9ab+31104t8b2+12465\
  9t7a3-13122t6a2b-263412t5ab2-1398528t4b3+1467477t3a3b-1414503t2a2b2-22543\
  8tab3+2088320b4
↦       _[8]=1705860yta3b+1516320ya2b2-269568t11b-729t9a2+1158624t8ab+87091\
  2t7b2+8748t6a3-2037798t5a2b+12301632t4ab2-1240320t3b3+1109376t2a3b+487676\
  7ta2b2+1731808ab3
↦       _[9]=12130560yt2ab2-1705860ya3b-425736t11a+642816t10b+1782405t8a2-1\
  403568t7ab-2612736t6b2-2956824t5a3+24555150t4a2b-35184456t3ab2+19255040t2\
  b3+6714252ta3b-4160403a2b2
↦       _[10]=3411720yt2a2b-1516320ytab2-4043520yb3+112266t10a+61560t9b-481\
  140t7a2-788292t6ab-221616t5b2+841995t4a3-5807700t3a2b-762534t2ab2-2104264\
  tb3-1043523a3b
↦       _[11]=171072yt3b2+413343yt2a3+393660yta2b+44712yab2+20412t9a+16038t\
  8b-107163t6a2-163296t5ab-160380t4b2+15309t3a3-817209t2a2b-329508tab2+3746\
  78b3
↦       _[12]=552yt3ab-448yt2b2-405yta3-228ya2b+70t11-300t8a-252t7b+525t5a2\
  -3384t4ab+630t3b2-295t2a3-1089ta2b-228ab2
↦       _[13]=2052yt3a2-648yt2ab-320ytb2+297ya3+50t10-312t7a-180t6b-309t4a2\
  -1440t3ab+450t2b2+571ta3+297a2b
↦       _[14]=66yt4b+81yt2a2+96ytab+14yb2+4t9-21t6a-54t5b+3t3a2-135t2ab-30t\
  b2+14a3
↦       _[15]=63yt4a-32yt3b+18yta2+5yab-3t8-21t5a+30t4b+24t2a2+33tab+5b2
↦       _[16]=10yt6+16yt3a+6yt2b+ya2-6t7+3t4a-10t3b+3ta2+ab
↦       _[17]=2y2b-12yt5-15yt2a-12ytb+7t6-5t3a+15t2b-2a2
↦        _[18]=3y2a+5yt4+4yta+3yb-3t5+3t2a-5tb
↦        _[19]=6y2t2-8yt3-ya+3t4-3ta-b
↦        _[20]=y3-3y2t+3yt2-t3+a
↦        _[21]=x+y-t
↦  [6]:
↦     [1]:
↦        1
↦     [2]:
↦        _[1]=b
↦        _[2]=a
↦        _[3]=t6
↦        _[4]=5yt4-3t5
↦        _[5]=6y2t2-8yt3+3t4
↦        _[6]=y3-3y2t+3yt2-t3
↦        _[7]=x+y-t
↦  [7]:
↦     [1]:
↦        a
↦     [2]:
↦        _[1]=b
↦        _[2]=a4
↦        _[3]=t6a3
↦        _[4]=5t8a2-28t5a3
↦        _[5]=14t10a-60t7a2+105t4a3
↦        _[6]=t12-4t9a+6t6a2-4t3a3
↦        _[7]=81ya3-14t10+60t7a-105t4a2+59ta3
↦        _[8]=81yt2a2+4t9-21t6a+3t3a2+14a3
↦        _[9]=21yt3a+6ya2-t7-7t4a+8ta2
↦        _[10]=12yt5+15yt2a-7t6+5t3a+2a2
↦        _[11]=3y2a+5yt4+4yta-3t5+3t2a
↦        _[12]=6y2t2-8yt3-ya+3t4-3ta
↦        _[13]=y3-3y2t+3yt2-t3+a
↦        _[14]=x+y-t
↦  [8]:
↦     [1]:
↦        1
↦     [2]:
↦        _[1]=b
↦        _[2]=a
↦        _[3]=t6
↦        _[4]=5yt4-3t5
↦        _[5]=6y2t2-8yt3+3t4
↦        _[6]=y3-3y2t+3yt2-t3
↦        _[7]=x+y-t
↦  [9]:
↦     [1]:
↦        b
↦     [2]:
↦        _[1]=a
↦        _[2]=b3
↦        _[3]=t6b2
↦        _[4]=5t9b-18t5b2
↦        _[5]=t12-3t8b+3t4b2
↦         _[6]=32yb2-5t9+18t5b-45tb2
↦         _[7]=32yt3b+3t8-30t4b-5b2
↦         _[8]=5yt4+3yb-3t5-5tb
↦         _[9]=10y2b-24ytb-t6+15t2b
↦         _[10]=6y2t2-8yt3+3t4-b
↦         _[11]=y3-3y2t+3yt2-t3
↦         _[12]=x+y-t
↦ [10]:
↦    [1]:
↦       ab
↦    [2]:
↦       _[1]=729a4+64b3
↦       _[2]=432t10b2-2187t9a3-1458t8a2b-2592t7ab2-2592t6b3+16038t5a3b+1263\
   6t4a2b2-13536t3ab3-3472t2b4+31077ta3b2+4758a2b3
↦       _[3]=5832t10ab+2592t9b2-19683t8a3-34992t7a2b-34992t6ab2-19008t5b3+1\
   70586t4a3b-182736t3a2b2-46872t2ab3-36832tb4+64233a3b2
↦       _[4]=6561t10a2+2916t9ab+1944t8b2-39366t7a3-39366t6a2b-21384t5ab2-16\
   848t4b3-205578t3a3b-52731t2a2b2-41436tab3-6344b4
↦       _[5]=648t11b-729t9a2-2916t8ab-2160t7b2+4374t6a3+8262t5a2b-28728t4ab\
   2+3816t3b3+20250t2a3b-13581ta2b2-3172ab3
↦       _[6]=2916t11a-648t10b-10935t8a2-5832t7ab+2160t6b2+13122t5a3-148230t\
   4a2b+37476t3ab2-2792t2b3-107730ta3b-21411a2b2
↦       _[7]=729t12-2916t9a-2187t8b+4374t6a2-34992t5ab+2187t4b2-2916t3a3-21\
   870t2a2b-8748tab2-793b3
↦       _[8]=112320yt2b3+568620yta3b+126360ya2b2-4374t9a2+12474t8ab+6336t7b\
   2+43011t6a3-54108t5a2b-80388t4ab2-75392t3b3-52407t2a3b-489222ta2b2-75062a\
   b3
↦       _[9]=505440yt2ab2-224640ytb3+568620ya3b-3888t10b+69255t8a2+51840t7a\
   b+6336t6b2-387828t5a3-475470t4a2b-217440t3ab2+51952t2b3-2481192ta3b-38060\
   1a2b2
↦       _[10]=3411720yt2a2b-1516320ytab2-336960yb3+112266t10a+61560t9b-4811\
   40t7a2-788292t6ab-221616t5b2+841995t4a3-5807700t3a2b-762534t2ab2+595576tb\
   3-1043523a3b
↦       _[11]=171072yt3b2+413343yt2a3+393660yta2b+44712yab2+20412t9a+16038t\
   8b-107163t6a2-163296t5ab-160380t4b2+15309t3a3-817209t2a2b-329508tab2-3300\
   2b3
↦       _[12]=552yt3ab-448yt2b2-405yta3-228ya2b+70t11-300t8a-252t7b+525t5a2\
   -3384t4ab+630t3b2-295t2a3-1089ta2b-228ab2
↦       _[13]=2052yt3a2-648yt2ab-320ytb2+297ya3+50t10-312t7a-180t6b-309t4a2\
   -1440t3ab+450t2b2+571ta3+297a2b
↦       _[14]=66yt4b+81yt2a2+96ytab+14yb2+4t9-21t6a-54t5b+3t3a2-135t2ab-30t\
   b2+14a3
↦       _[15]=63yt4a-32yt3b+18yta2+5yab-3t8-21t5a+30t4b+24t2a2+33tab+5b2
↦       _[16]=10yt6+16yt3a+6yt2b+ya2-6t7+3t4a-10t3b+3ta2+ab
↦       _[17]=2y2b-12yt5-15yt2a-12ytb+7t6-5t3a+15t2b-2a2
↦       _[18]=3y2a+5yt4+4yta+3yb-3t5+3t2a-5tb
↦       _[19]=6y2t2-8yt3-ya+3t4-3ta-b
↦       _[20]=y3-3y2t+3yt2-t3+a
↦       _[21]=x+y-t
↦ [11]:
↦    [1]:
↦       1
↦    [2]:
```

```
↦          _[1]=b
↦          _[2]=a
↦          _[3]=t6
↦          _[4]=5yt4-3t5
↦          _[5]=6y2t2-8yt3+3t4
↦          _[6]=y3-3y2t+3yt2-t3
↦          _[7]=x+y-t
↦  [12]:
↦     [1]:
↦        a
↦     [2]:
↦        _[1]=b
↦        _[2]=a4
↦        _[3]=t6a3
↦        _[4]=5t8a2-28t5a3
↦        _[5]=14t10a-60t7a2+105t4a3
↦        _[6]=t12-4t9a+6t6a2-4t3a3
↦        _[7]=81ya3-14t10+60t7a-105t4a2+59ta3
↦        _[8]=81yt2a2+4t9-21t6a+3t3a2+14a3
↦        _[9]=21yt3a+6ya2-t7-7t4a+8ta2
↦        _[10]=12yt5+15yt2a-7t6+5t3a+2a2
↦        _[11]=3y2a+5yt4+4yta-3t5+3t2a
↦        _[12]=6y2t2-8yt3-ya+3t4-3ta
↦        _[13]=y3-3y2t+3yt2-t3+a
↦        _[14]=x+y-t
↦  [13]:
↦     [1]:
↦        1
↦     [2]:
↦        _[1]=b
↦        _[2]=a
↦        _[3]=t6
↦        _[4]=5yt4-3t5
↦        _[5]=6y2t2-8yt3+3t4
↦        _[6]=y3-3y2t+3yt2-t3
↦        _[7]=x+y-t
↦  [14]:
↦     [1]:
↦        b
↦     [2]:
↦        _[1]=a
↦        _[2]=b3
↦        _[3]=t6b2
↦        _[4]=5t9b-18t5b2
↦        _[5]=t12-3t8b+3t4b2
↦        _[6]=32yb2-5t9+18t5b-45tb2
↦        _[7]=32yt3b+3t8-30t4b-5b2
↦        _[8]=5yt4+3yb-3t5-5tb
↦        _[9]=10y2b-24ytb-t6+15t2b
↦        _[10]=6y2t2-8yt3+3t4-b
↦        _[11]=y3-3y2t+3yt2-t3
↦        _[12]=x+y-t
↦  [15]:
```

```
↦    [1]:
↦       ab
↦    [2]:
↦       _[1]=16767a4+5632b3
↦       _[2]=16767t12-67068t9a-50301t8b+100602t6a2-804816t5ab+50301t4b2-670\
  68t3a3-503010t2a2b-201204tab2-22399b3
↦       _[3]=32348160yb4+27766152t11a2-2146176t10ab+476928t9b2-114535377t8a\
  3-78067152t7a2b+2861568t6ab2-63272448t5b3-1314163926t4a3b+210548808t3a2b2\
  +27688320t2ab3+228423424tb4-183555801a3b2
↦       _[4]=2274480ya2b3-655776t11b2-150903t10a3+33534t9a2b+2705076t8ab2+1\
  843776t7b3+201204t6a3b-4448844t5a2b2+31037688t4ab3-4972704t3b4+1946835t2a\
  3b2+16061022ta2b3+4335188ab4
↦       _[5]=10235160ya3b2-2950992t11ab+228096t10b2+150903t9a3+12172842t8a2\
  b+8296992t7ab2-304128t6b3-20019798t5a3b+139669596t4a2b2-22377168t3ab3-294\
  2720t2b4+72274599ta3b2+19508346a2b3
↦       _[6]=1797120ytb3+13078260ya3b-3889944t11a+317952t10b+15844815t8a2+1\
  0760688t7ab-794880t6b2-24546888t5a3+185536170t4a2b-29127384t3ab2-6614272t\
  2b3+100664100ta3b+26988039a2b2
↦       _[7]=26156520yta2b2+2021760yab3+251505t10a2+972486t9ab+983664t8b2-2\
  565351t7a3-5734314t6a2b-9009468t5ab2-6972768t4b3+5382207t3a3b-39810447t2a\
  2b2-26365962tab3-6221072b4
↦       _[8]=1705860yta3b+379080ya2b2-71280t11b+67068t9a2+358182t8ab+256608\
  t7b2-352107t6a3-1071144t5a2b+2918916t4ab2-658416t3b3-2384721t2a3b+26244ta\
  2b2+65494ab3
↦       _[9]=6065280yt2ab2+42646500ya3b-12206376t11a+1168992t10b+50049495t8\
  a2+33199632t7ab-3250368t6b2-78871968t5a3+581360490t4a2b-102330216t3ab2-17\
  630624t2b3+303270156ta3b+80747199a2b2
↦       _[10]=78469560yt2a2b-34875360ytab2-4043520yb3+2582118t10a+1415880t9\
  b-11066220t7a2-18130716t6ab-5097168t5b2+19365885t4a3-133577100t3a2b-17538\
  282t2ab2+16398088tb3-24001029a3b
↦       _[11]=3934656yt3b2+9506889yt2a3+9054180yta2b+1028376yab2+469476t9a+\
  368874t8b-2464749t6a2-3755808t5ab-3688740t4b2+352107t3a3-18795807t2a2b-75\
  78684tab2-1166726b3
↦       _[12]=552yt3ab-448yt2b2-405yta3-228ya2b+70t11-300t8a-252t7b+525t5a2\
  -3384t4ab+630t3b2-295t2a3-1089ta2b-228ab2
↦       _[13]=2052yt3a2-648yt2ab-320ytb2+297ya3+50t10-312t7a-180t6b-309t4a2\
  -1440t3ab+450t2b2+571ta3+297a2b
↦       _[14]=66yt4b+81yt2a2+96ytab+14yb2+4t9-21t6a-54t5b+3t3a2-135t2ab-30t\
  b2+14a3
↦       _[15]=63yt4a-32yt3b+18yta2+5yab-3t8-21t5a+30t4b+24t2a2+33tab+5b2
↦       _[16]=10yt6+16yt3a+6yt2b+ya2-6t7+3t4a-10t3b+3ta2+ab
↦       _[17]=2y2b-12yt5-15yt2a-12ytb+7t6-5t3a+15t2b-2a2
↦       _[18]=3y2a+5yt4+4yta+3yb-3t5+3t2a-5tb
↦       _[19]=6y2t2-8yt3-ya+3t4-3ta-b
↦       _[20]=y3-3y2t+3yt2-t3+a
↦       _[21]=x+y-t
↦ [16]:
↦    [1]:
↦       1
↦    [2]:
↦       _[1]=b
↦       _[2]=a
↦       _[3]=t6
↦          _[4]=5yt4-3t5
↦          _[5]=6y2t2-8yt3+3t4
↦          _[6]=y3-3y2t+3yt2-t3
↦          _[7]=x+y-t
↦    [17]:
↦       [1]:
↦          a
↦       [2]:
↦          _[1]=b
↦          _[2]=t12-4t9a+6t6a2-4t3a3+a4
↦          _[3]=81ya3-14t10+60t7a-105t4a2+59ta3
↦          _[4]=81yt2a2+4t9-21t6a+3t3a2+14a3
↦          _[5]=21yt3a+6ya2-t7-7t4a+8ta2
↦          _[6]=12yt5+15yt2a-7t6+5t3a+2a2
↦          _[7]=3y2a+5yt4+4yta-3t5+3t2a
↦          _[8]=6y2t2-8yt3-ya+3t4-3ta
↦          _[9]=y3-3y2t+3yt2-t3+a
↦          _[10]=x+y-t
↦    [18]:
↦       [1]:
↦          1
↦       [2]:
↦          _[1]=b
↦          _[2]=a
↦          _[3]=t6
↦          _[4]=5yt4-3t5
↦          _[5]=6y2t2-8yt3+3t4
↦          _[6]=y3-3y2t+3yt2-t3
↦          _[7]=x+y-t
↦    [19]:
↦       [1]:
↦          b
↦       [2]:
↦          _[1]=a
↦          _[2]=t12-3t8b+3t4b2-b3
↦          _[3]=32yb2-5t9+18t5b-45tb2
↦          _[4]=32yt3b+3t8-30t4b-5b2
↦          _[5]=5yt4+3yb-3t5-5tb
↦          _[6]=10y2b-24ytb-t6+15t2b
↦          _[7]=6y2t2-8yt3+3t4-b
↦          _[8]=y3-3y2t+3yt2-t3
↦          _[9]=x+y-t
↦    [20]:
↦       [1]:
↦          -8910671247a13b+46290636864a9b4+20949663744a5b7+1476395008ab10
↦       [2]:
↦          _[1]=t12-4t9a-3t8b+6t6a2-48t5ab+3t4b2-4t3a3-30t2a2b-12tab2+a4-b3
↦          _[2]=531441ya8-2939328ya4b3-262144yb6+673920t11a2b2-91854t10a5-8294\
     4t10ab3+87480t9a4b+40960t9b4-2779920t8a3b2+393660t7a6-1762560t7a2b3-78732\
     t6a5b+43008t6ab4+4132944t5a4b2-147456t5b5-688905t4a7-32127840t4a3b3+48551\
     40t3a6b+6741120t3a2b4-7735014t2a5b2-1926144t2ab5+387099ta8-15277896ta4b3+\
     368640tb6+1336257a7b-4006288a3b4
↦          _[3]=6561yta5+576ytab3+5832ya4b+512yb4-1134t11a2+72t10ab-80t9b2+486\
     0t8a3+3348t7a2b+240t6ab2-8505t5a4+288t5b3+52380t4a3b-8934t3a2b2+4779t2a5+\
     2952t2ab3+20745ta4b-720tb4+6344a3b2
↦        _[4]=373248yta4b2+32768ytb5+59049ya7+5184ya3b3+10368t11ab2-10206t10\
     a4-5120t10b3+9720t9a3b-32400t8a2b2+43740t7a5-5376t7ab3-8748t6a4b+18432t6b\
     4-24624t5a3b2-76545t4a6-589920t4a2b3+539460t3a5b+240768t3ab4-587574t2a4b2\
     -46080t2b5+43011ta7-517384ta3b3+148473a6b-84240a2b4
↦        _[5]=9360yt2ab2+13851yta4-2944ytb3+10530ya3b-2394t11a+460t10b+10260\
     t8a2+5748t7ab-1656t6b2-17955t5a3+112890t4a2b-34794t3ab2+10089t2a4+4140t2b\
     3+42497ta3b+10530a2b2
↦        _[6]=42120yt2a2b-18720ytab2-8019ya4-4864yb3+1386t10a+760t9b-5940t7a\
     2-9732t6ab-2736t5b2+10395t4a3-71700t3a2b-9414t2ab2-5841ta4+6840tb3-12883a\
     3b
↦        _[7]=266240yt2b4+1347840yta3b2+150903ya6+312768ya2b3-41600t11b2-260\
     82t10a3+24840t9a2b+209040t8ab2+111780t7a4+149760t7b3-22356t6a3b-573648t5a\
     2b2-195615t4a5+1703520t4ab3+1378620t3a4b-374400t3b4-1214602t2a3b2+109917t\
     a6-113400ta2b3+379431a5b+84240ab4
↦        _[8]=16767yt2a4+5632yt2b3+21060yta3b+4680ya2b2-880t11b+828t9a2+4422\
     t8ab+3168t7b2-4347t6a3-13224t5a2b+36036t4ab2+621t3a4-7920t3b3-29441t2a3b+\
     324ta2b2+2898a5+1782ab3
↦        _[9]=704yt3b2+1701yt2a3+1620yta2b+184yab2+84t9a+66t8b-441t6a2-672t5\
     ab-660t4b2+63t3a3-3363t2a2b-1356tab2+294a4-110b3
↦        _[10]=552yt3ab-448yt2b2-405yta3-228ya2b+70t11-300t8a-252t7b+525t5a2\
     -3384t4ab+630t3b2-295t2a3-1089ta2b-228ab2
↦        _[11]=2052yt3a2-648yt2ab-320ytb2+297ya3+50t10-312t7a-180t6b-309t4a2\
     -1440t3ab+450t2b2+571ta3+297a2b
↦        _[12]=66yt4b+81yt2a2+96ytab+14yb2+4t9-21t6a-54t5b+3t3a2-135t2ab-30t\
     b2+14a3
↦        _[13]=63yt4a-32yt3b+18yta2+5yab-3t8-21t5a+30t4b+24t2a2+33tab+5b2
↦        _[14]=10yt6+16yt3a+6yt2b+ya2-6t7+3t4a-10t3b+3ta2+ab
↦        _[15]=2y2b-12yt5-15yt2a-12ytb+7t6-5t3a+15t2b-2a2
↦        _[16]=3y2a+5yt4+4yta+3yb-3t5+3t2a-5tb
↦        _[17]=6y2t2-8yt3-ya+3t4-3ta-b
↦        _[18]=y3-3y2t+3yt2-t3+a
↦        _[19]=x+y-t
```

D.2.2.2 base2str

Procedure from library `compregb.lib` (see Section D.2.2 [compregb_lib], page 797).

D.2.3 general_lib

Library: general.lib

Purpose: Elementary Computations of General Type

Procedures:

D.2.3.1 A_Z

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: A_Z("a",n); a any letter, n integer (-26<= n <=26, !=0)

Return: string of n small (if a is small) or capital (if a is capital) letters, comma separated, beginning with a, in alphabetical order (or reverse alphabetical order if n<0)

Example:

```
LIB "general.lib";
A_Z("c",5);
↦ c,d,e,f,g
A_Z("Z",-5);
↦ Z,Y,X,W,V
ring R = create_ring("(0,"+A_Z("A",6)+")", "("+A_Z("a",10)+")", "dp");
R;
↦ // coefficients: QQ(A, B, C, D, E, F)
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    a b c d e f g h i j
↦ //        block   2 : ordering C
```

D.2.3.2 A_Z_L

Procedure from library general.lib (see Section D.2.3 [general_lib], page 805).

Usage: A_Z_L("a",n); a any letter, n integer (-26<= n <=26, !=0)

Return: list of n small (if a is small) or capital (if a is capital) letters, beginning with a, in alphabetical
order (or reverse alphabetical order if n<0)

Example:

```
LIB "general.lib";
A_Z_L("c",5);
↦ [1]:
↦    c
↦ [2]:
↦    d
↦ [3]:
↦    e
↦ [4]:
↦    f
↦ [5]:
↦    g
A_Z_L("Z",-5);
↦ [1]:
↦    Z
↦ [2]:
↦    Y
↦ [3]:
↦    X
↦ [4]:
↦    W
↦ [5]:
↦    V
ring r;
list L=list(0,A_Z_L("A",6),list(list("dp",1:6),list("C",0)),ideal(0));
ring R=ring(L);
R;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names     A B C D E F
↦ //        block   2 : ordering C
```

D.2.3.3 ASCII

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: ASCII([n,m]); n,m= integers (32 <= n <= m <= 126)

Return: string of printable ASCII characters (no native language support)
ASCII(): string of all ASCII characters with its numbers,
ASCII(n): n-th ASCII character
ASCII(n,m): n-th up to m-th ASCII character (inclusive)

Example:

```
LIB "general.lib";
ASCII();"";
↦      !    "    #    $    %    &    ’    (    )    *    +    ,    -    .
↦ 32   33   34   35   36   37   38   39   40   41   42   43   44   45   46
↦ /    0    1    2    3    4    5    6    7    8    9    :    ;    <    =
↦ 47   48   49   50   51   52   53   54   55   56   57   58   59   60   61
↦ >    ?    @    A    B    C    D    E    F    G    H    I    J    K    L
↦ 62   63   64   65   66   67   68   69   70   71   72   73   74   75   76
↦ M    N    O    P    Q    R    S    T    U    V    W    X    Y    Z    [
↦ 77   78   79   80   81   82   83   84   85   86   87   88   89   90   91
↦ \    ]    ^    _    ‘    a    b    c    d    e    f    g    h    i    j
↦ 92   93   94   95   96   97   98   99  100  101  102  103  104  105  10
↦ k    l    m    n    o    p    q    r    s    t    u    v    w    x    y
↦ 107  108  109  110  111  112  113  114  115  116  117  118  119  120  121
↦ z    {    |    }    ~
↦ 122  123  124  125  126
↦
ASCII(42);
↦ *
ASCII(32,126);
↦  !"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefgh\
   ijklmnopqrstuvwxyz{|}~
```

D.2.3.4 absValue

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: absValue(c); c int, number or poly

Return: absValue(c); the absolute value of c

Note: absValue(c)=c if c>=0; absValue=-c if c<=0.
So the function can be applied to any type, for which comparison operators are defined.

Example:

```
LIB "general.lib";
ring r1 = 0,x,dp;
absValue(-2002);
↦ 2002
```

```
poly f=-4;
absValue(f);
↦ 4
```

D.2.3.5 binomial

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: binomial(n,k); n,k integers

Return: binomial(n,k); binomial coefficient n choose k

Example:

```
LIB "general.lib";
binomial(200,100);"";                 //type bigint
↦ 90548514656103281165404177077484163874504589675413336841320
↦ 
int n,k = 200,100;
bigint b1 = binomial(n,k);
ring r = 0,x,dp;
poly b2 = coeffs((x+1)^n,x)[k+1,1];  //coefficient of x^k in (x+1)^n
b1-b2;                               //b1 and b2 should coincide
↦ 0
```

D.2.3.6 deleteSublist

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: deleteSublist(v,l); intvec v; list l
where the entries of the integer vector v correspond to the positions of the elements to be deleted

Return: list without the deleted elements

Example:

```
LIB "general.lib";
list l=1,2,3,4,5;
intvec v=1,3,4;
l=deleteSublist(v,l);
l;
↦ [1]:
↦    2
↦ [2]:
↦    5
```

D.2.3.7 factorial

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: factorial(n); n integer

Return: factorial(n): n! of type bigint.

Example:

```
LIB "general.lib";
factorial(37);
↦ 13763753091226345046315979581580902400000000
```

See also: Section 5.1.118 [prime], page 241.

D.2.3.8 fibonacci

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: fibonacci(n); n,p integers

Return: fibonacci(n): nth Fibonacci number, f(0)=f(1)=1, f(i+1)=f(i-1)+f(i)
- computed in characteristic 0, of type bigint

Example:

```
LIB "general.lib";
fibonacci(42);
↦ 267914296
```

See also: Section 5.1.118 [prime], page 241.

D.2.3.9 kmemory

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: kmemory([n,[v]]); n,v integers

Return: memory in kilobyte of type bigint
n=0: memory used by active variables (same as no parameters)
n=1: total memory allocated by Singular

Display: detailed information about allocated and used memory if v!=0

Note: kmemory uses internal function 'memory' to compute kilobyte, and is the same as 'memory' for n!=0,1,2

Example:

```
LIB "general.lib";
kmemory();
↦ 155
kmemory(1,1);
↦ // total memory allocated, at the moment, by SINGULAR (kilobyte):
↦ 2100
```

D.2.3.10 killall

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: killall(); (no parameter)
killall("type_name");
killall("not", "type_name");

Return: killall(); kills all user-defined variables except loaded procedures, no return value.
- killall("type_name"); kills all user-defined variables, of type "type_name"
- killall("not", "type_name"); kills all user-defined variables, except those of type "type_name" and except loaded procedures
- killall("not", "name_1", "name_2", ...); kills all user-defined variables, except those of name "name_i" and except loaded procedures

Note: killall should never be used inside a procedure

Example:

```
LIB "general.lib";
ring rtest; ideal i=x,y,z; string str="hi"; int j = 3;
export rtest,i,str,j;       //this makes the local variables global
listvar();
↦ // j                              [0]  int 3
↦ // str                            [0]  string hi
↦ // rtest                          [0]  *ring
↦ //      i                              [0]  ideal, 3 generator(s)
killall("ring");            // kills all rings
↦ // ** killing the basering for level 0
listvar();
↦ // j                              [0]  int 3
↦ // str                            [0]  string hi
killall("not", "int");      // kills all variables except int's (and procs)
↦ // ** cannot kill `Singmathic`
listvar();
↦ // j                              [0]  int 3
↦ // str                            [0]  string hi
killall();                  // kills all vars except loaded procs
listvar();
↦ // j                              [0]  int 3
```

D.2.3.11 number_e

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: number_e(n); n integer

Return: Euler number e=exp(1) up to n decimal digits (no rounding)
- of type string if no basering of char 0 is defined
- of type number if a basering of char 0 is defined

Display: decimal format of e if printlevel > 0 (default:printlevel=0)

Note: procedure uses algorithm of A.H.J. Sale

Example:

```
LIB "general.lib";
number_e(30);"";
↦ 2.71828182845904523536028747135
↦
ring R = 0,t,lp;
number e = number_e(30);
e;
↦ 13591409142295226176801437356763/5000000000000000000000000000000
```

D.2.3.12 number_pi

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: number_pi(n); n positive integer

Return: pi (area of unit circle) up to n decimal digits (no rounding)
- of type string if no basering of char 0 is defined,
- of type number, if a basering of char 0 is defined

Display: decimal format of pi if printlevel > 0 (default:printlevel=0)

Note: procedure uses algorithm of S. Rabinowitz

Example:

```
LIB "general.lib";
number_pi(11);"";
↦ 3.1415926535
↦
ring r = (real,10),t,dp;
number pi = number_pi(11); pi;
↦ 3.141592654
```

D.2.3.13 primes

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: primes(n,m); n,m integers

Return: intvec, consisting of all primes p, prime(n)<=p<=m, in increasing order if n<=m, resp. prime(m)<=p<=n, in decreasing order if m<n.

Note: prime(n); returns the biggest prime number <= min(n,32003) if n>=2, else 2

Example:

```
LIB "general.lib";
primes(50,100);"";
↦ 47,53,59,61,67,71,73,79,83,89,97
↦
intvec v = primes(37,1); v;
↦ 37,31,29,23,19,17,13,11,7,5,3,2
```

D.2.3.14 product

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: product(id[,v]); id ideal/vector/module/matrix/intvec/intmat/list, v intvec (default: v=1..number of entries of id)

Assume: list members can be multiplied.

Return: The product of all entries of id [with index given by v] of type depending on the entries of id.

Note: If id is not a list, id is treated as a list of polys resp. integers. A module m is identified with the corresponding matrix M (columns of M generate m).
If v is outside the range of id, we have the empty product and the result will be 1 (of type int).

Example:

```
LIB "general.lib";
ring r= 0,(x,y,z),dp;
ideal m = maxideal(1);
product(m);
↦ xyz
product(m[2..3]);
↦ yz
matrix M[2][3] = 1,x,2,y,3,z;
product(M);
```

```
↦ 6xyz
intvec v=2,4,6;
product(M,v);
↦ xyz
intvec iv = 1,2,3,4,5,6,7,8,9;
v=1..5,7,9;
product(iv,v);
↦ 7560
intmat A[2][3] = 1,1,1,2,2,2;
product(A,3..5);
↦ 4
```

D.2.3.15 sort

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: sort(id[,v,o,n]); id = ideal/module/intvec/list
sort may be called with 1, 2 or 3 arguments in the following way:
sort(id[,v,n]); v=intvec of positive integers, n=integer,
sort(id[,o,n]); o=string (any allowed ordstr of a ring), n=integer

Return: a list l of two elements:

l[1]: object of same type as input but sorted in the following way:
- if id=ideal/module: generators of id are sorted w.r.t. intvec v (id[v[1]] becomes 1-st, id[v[2]] 2-nd element, etc.). If no v is present, id is sorted w.r.t. ordering o (if o is given) or w.r.t. actual monomial ordering (if no o is given):
NOTE: generators with SMALLER(!) leading term come FIRST
(e.g. sort(id); sorts backwards to actual monomial ordering)
- if id=list or intvec: sorted w.r.t. < (indep. of other arguments)
- if n!=0 the ordering is inverse, i.e. w.r.t. v(size(v)..1)
default: n=0

l[2]: intvec, describing the permutation of the input (hence l[2]=v if v is given (with positive integers))

Note: If v is given id may be any simply indexed object (e.g. any list or string); if v[i]<0 and i<=size(id) v[i] is set internally to i; entries of v must be pairwise distinct to get a permutation id. Zero generators of ideal/module are deleted
If 'o' is given, the input is sorted by considering leading terms w.r.t. the new ring ordering given by 'o'

Example:

```
LIB "general.lib";
ring r0 = 0,(x,y,z,t),lp;
ideal i = x3,z3,xyz;
sort(i);             //sorts using lex ordering, smaller polys come first
↦ [1]:
↦    _[1]=z3
↦    _[2]=xyz
↦    _[3]=x3
↦ [2]:
↦    2,3,1
sort(i,3..1);
↦ [1]:
```

```
↦    _[1]=xyz
↦    _[2]=z3
↦    _[3]=x3
↦ [2]:
↦    3,2,1
sort(i,"ls")[1];     //sort w.r.t. negative lex ordering
↦ _[1]=x3
↦ _[2]=xyz
↦ _[3]=z3
intvec v =1,10..5,2..4;v;
↦ 1,10,9,8,7,6,5,2,3,4
sort(v)[1];          // sort v lexicographically
↦ 1,2,3,4,5,6,7,8,9,10
sort(v,"Dp",1)[1];   // sort v w.r.t (total sum, reverse lex)
↦ 1,2,3,4,5,6,7,8,9,10
// Note that in general: lead(sort(M)) != sort(lead(M)), e.g:
module M = [0, 1, 1, 0], [1, 0, 0, 1]; M;
↦ M[1]=gen(3)+gen(2)
↦ M[2]=gen(4)+gen(1)
sort(lead(M), "c, dp")[1];
↦ _[1]=gen(4)
↦ _[2]=gen(3)
lead(sort(M, "c, dp")[1]);
↦ _[1]=gen(3)
↦ _[2]=gen(4)
// In order to sort M wrt a NEW ordering by considering OLD leading
// terms use one of the following equivalent commands:
module( M[ sort(lead(M), "c,dp")[2] ] );
↦ _[1]=gen(4)+gen(1)
↦ _[2]=gen(3)+gen(2)
sort( M, sort(lead(M), "c,dp")[2] )[1];
↦ _[1]=gen(4)+gen(1)
↦ _[2]=gen(3)+gen(2)
// BUG: Please, don't use this sort for integer vectors or lists
// with them if there can be negative integers!
// TODO: for some HiWi
sort(3..-3)[1];
↦ -3,-2,-1,0,1,2,3
sort(list(-v, v))[1];
↦ [1]:
↦    -1,-10,-9,-8,-7,-6,-5,-2,-3,-4
↦ [2]:
↦    1,10,9,8,7,6,5,2,3,4
```

D.2.3.16 sum

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: sum(id[,v]); id ideal/vector/module/matrix/intvec/intmat/list, v intvec (default: v=1..number of entries of id)

Assume: list members can be added.

Return: The sum of all entries of id [with index given by v] of type depending on the entries of id.

Note: If id is not a list, id is treated as a list of polys resp. integers. A module m is identified with the corresponding matrix M (columns of M generate m).
If v is outside the range of id, we have the empty sum and the result will be 0 (of type int).

Example:

```
LIB "general.lib";
ring r1 = 0,(x,y,z),dp;
vector pv = [xy,xz,yz,x2,y2,z2];
sum(pv);
↦ x2+xy+y2+xz+yz+z2
sum(pv,2..5);
↦ x2+y2+xz+yz
matrix M[2][3] = 1,x,2,y,3,z;
intvec w=2,4,6;
sum(M,w);
↦ x+y+z
intvec iv = 1,2,3,4,5,6,7,8,9;
sum(iv,2..4);
↦ 9
iv = intvec(1..100);
sum(iv);
↦ 5050
ring r2 = 0,(x(1..10)),dp;
sum(x(3..7),intvec(1,3,5));
↦ x(3)+x(5)+x(7)
```

D.2.3.17 watchdog

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Return: Result of cmd, if the result can be computed in i seconds. Otherwise the computation is interrupted after i seconds, the string "Killed" is returned and the global variable 'watchdog_interrupt' is defined.

Note: * the current basering should not be watchdog_rneu, since watchdog_rneu will be killed
* if there are variable names of the structure x(i) all polynomials have to be put into eval(...) in order to be interpreted correctly
* a second Singular process is started by this procedure

Example:

```
LIB "general.lib";
proc sleep(int s) {return(system("sh","sleep "+string(s)));}
watchdog(1,"sleep(5)");
↦ Killed
watchdog(10,"sleep(5)");
↦ 0
```

D.2.3.18 primecoeffs

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: primecoeffs(J[,p]); J any type which can be converted to a matrix e.g. ideal, matrix, vector, module, int, intvec
p = integer

Compute: primefactors <= p of coeffs of J (default p = 32003)

Return: a list, say l, of two intvectors:
l[1] : the different primefactors of all coefficients of J
l[2] : the different remaining factors

Example:

```
LIB "general.lib";
primecoeffs(intvec(7*8*121,7*8));"";
↦ [1]:
↦    2,7,11
↦ [2]:
↦    1
↦
ring r = 0,(b,c,t),dp;
ideal I = -13b6c3t+4b5c4t,-10b4c2t-5b4ct2;
primecoeffs(I);
↦ [1]:
↦    2,5,13
↦ [2]:
↦    _[1]=-1
↦    _[2]=1
```

D.2.3.19 timeStd

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: timeStd(i,d), i ideal, d integer

Return: std(i) if the standard basis computation finished after d-1 seconds and i otherwise

Example:

```
LIB "general.lib";
ring r=32003,(a,b,c,d,e),dp;
int n=7;
ideal i=
a^n-b^n,
b^n-c^n,
c^n-d^n,
d^n-e^n,
a^(n-1)*b+b^(n-1)*c+c^(n-1)*d+d^(n-1)*e+e^(n-1)*a;
def i1=timeStd(i,1);
def i2=timeStd(i,20);
listvar();
↦ // n                           [0]  int 7
↦ // r                           [0]  *ring
↦ //      i2                          [0]  ideal (SB), 746 generator(s)
↦ //      i1                          [0]  ideal, 5 generator(s)
↦ //      i                           [0]  ideal, 5 generator(s)
```

D.2.3.20 timeFactorize

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: timeFactorize(p,d); poly p , integer d

Return: factorize(p) if the factorization finished after d-1
seconds otherwise f is considered to be irreducible

Example:

```
LIB "general.lib";
ring r=0,(x,y),dp;
poly p=((x2+y3)^2+xy6)*((x3+y2)^2+x10y);
p=p^2;
timeFactorize(p,2);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       x22y14+2x21y14+4x23y11+x20y14+2x25y8+4x22y11+6x24y8+4x26y5+2x18y13+\
   x28y2+4x17y13+4x15y15+8x19y10+2x16y13+8x14y15+2x12y17+4x21y7+8x18y10+16x1\
   6y12+4x13y15+4x11y17+12x20y7+8x18y9+16x15y12+8x13y14+2x10y17+8x22y4+24x17\
   y9+4x15y11+x14y12+8x12y14+2x24y+16x19y6+12x14y11+2x13y12+4x11y14+4x21y3+8\
   x16y8+4x15y9+x12y12+8x10y14+6x8y16+2x18y5+2x17y6+4x14y9+16x12y11+4x9y14+1\
   2x7y16+4x5y18+6x16y6+8x14y8+16x11y11+24x9y13+6x6y16+8x4y18+x2y20+4x18y3+2\
   4x13y8+12x11y10+24x8y13+16x6y15+4x3y18+2xy20+x20+16x15y5+36x10y10+8x8y12+\
   16x5y15+4x3y17+y20+4x17y2+24x12y7+24x7y12+2x5y14+4x2y17+6x14y4+16x9y9+6x4\
   y14+4x11y6+4x6y11+x8y8
↦ [2]:
↦    1,1
//timeFactorize(p,20);
```

D.2.3.21 factorH

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 805).

Usage: factorH(p) p poly

Return: factorize(p)

Note: changes variables to make the last variable the principal one in the multivariate factorization and factorizes then the polynomial

Example:

```
LIB "general.lib";
system("random",992851144);
ring r=32003,(x,y,z,w,t),lp;
poly p=y2w9+yz7t-yz5w4-z2w4t4-w8t3;
factorize(p);  //fast
↦ [1]:
↦    _[1]=-1
↦    _[2]=-y2w9-yz7t+yz5w4+z2w4t4+w8t3
↦ [2]:
↦    1,1
system("random",992851262);
//factorize(p);  //slow
system("random",992851262);
factorH(p);
↦ [1]:
↦    _[1]=1
↦    _[2]=y2w9+yz7t-yz5w4-z2w4t4-w8t3
↦ [2]:
↦    1,1
```

D.2.4 grobcov_lib

Library: grobcov.lib

Purpose:

"Groebner Cover for parametric ideals.", Comprehensive Groebner Systems, Groebner Cover, Canonical Forms, Parametric Polynomial Systems, Automatic Deduction of Geometric Theorems, Dynamic Geometry, Loci, Envelope, Constructible sets. See: A. Montes A, M. Wibmer, "Groebner Bases for Polynomial Systems with parameters", Journal of Symbolic Computation 45 (2010) 1391-1425. (https://www.mat.upc.edu//en/people/antonio.montes/).

Important:

Recently published book:

A. Montes. " The Groebner Cover":
Springer, Algorithms and Computation in Mathematics 27 (2019) ISSN 1431-1550
ISBN 978-3-030-03903-5
ISBN 978-3-030-03904-2 (e-Book)
Springer Nature Switzerland AG 2018

https://www.springer.com/gp/book/9783030039035

The book can also be used as a user manual of all
the routines included in this library.
It defines and proves all the theoretic results used in the library, and shows examples of all the routines. There are many previous papers related to the subject, and the book actualices all the contents.

Authors: Antonio Montes (Universitat Politecnica de Catalunya), Hans Schoenemann (Technische Universitaet Kaiserslautern).

Overview: In 2010, the library was designed to contain
Montes-Wibmer's algorithm for computing the
Canonical Groebner Cover of a parametric ideal.
The central routine is grobcov.
Given a parametric ideal, grobcov outputs
its Canonical Groebner Cover, consisting of a set
of triplets of (lpp, basis, segment). The basis
(after normalization) is the reduced Groebner basis for each point of the segment. The segments
are disjoint, locally closed and correspond to
constant lpp (leading power product) of the basis,
and are represented in canonical representation.
The segments cover the whole parameter space.
The output is canonical, it only depends on the
given parametric ideal and the monomial order,
because the segments have
different lpph of the homogenized system.
This is much more than a simple Comprehensive
Groebner System. The algorithm grobcov allows
options to solve partially the problem when the

whole automatic algorithm does not finish in
reasonable time. Its existence was proved for the
first time by Michael Wibmer "Groebner bases for
families of affine or projective schemes",
JSC, 42,803-834 (2007).

grobcov uses a first algorithm cgsdr that outputs a disjoint reduced Comprehensive Groebner System
with constant lpp. For this purpose, in this library, the implemented algorithm is Kapur-Sun-Wang
algorithm, because it is actually the most efficient algorithm known for this purpose.
D. Kapur, Y. Sun, and D.K. Wang "A New Algorithm
for Computing Comprehensive Groebner Systems".
Proceedings of ISSAC'2010, ACM Press, (2010), 29-36.

The library has evolved to include new applications of the Groebner Cover, and new theoretical developments have been done.

A routine locus has been included to compute
loci of points, and determining the taxonomy of the components.
Additional routines to transform the output to string (locusdg, locusto) are also included and used in the Dynamic Geometry software GeoGebra. They were
described in:
M.A. Abanades, F. Botana, A. Montes, T. Recio:
"An Algebraic Taxonomy for Locus Computation in
Dynamic Geometry".
Computer-Aided Design 56 (2014) 22-33.

Routines for determining the generalized envelope of a family of hypersurfaces (envelop, AssocTanToEnv,
FamElemsToEnvCompPoints) are also included.

It also includes procedures for
Automatic Deduction of Geometric Theorems (ADGT).

The actual version also includes a
routine (ConsLevels) for computing the canonical form of a constructible set, given as a union of locally closed sets. It determines the canonical
levels of a constructible set. It is described in:
J.M. Brunat, A. Montes, "Computing the canonical
representation of constructible sets".
Math. Comput. Sci. (2016) 19: 165-178.
A complementary routine Levels transforms the output of ConsLevels into the proper locally closed sets
forming the levels of the constructible.
Antoher complementary routine Grob1Levels
has been included to select the locally closed sets of the segments of the grobcov that correspond to basis different from 1, add them together and return
the canonical form of this constructible set.
More recently (2019) given two locally closed sets
in canonical form the new routine DifConsLCSets
determines a set of locally closed sets equivalent to the difference them. The description of the
routine is submitted to the Journal of Symbolic Computation. This routine can be also used internally by ADGT

with the option "neg",1 . With this option
DifConsLCSets is used for the negative
hypothesis and thesis in ADGT.

The last version N11 (2021) has improved the routines for locus and allows to determine a parametric locus.

This version was finished on 1/2/2021,

Notations: Before calling any routine of the library grobcov, the user must define the ideal Q[a][x], and all the input polynomials and ideals defined on it.
Internally the routines define and use also other
ideals: Q[a], Q[x,a] and so on.

Procedures: See also: Section D.2.2 [compregb_lib], page 797.

D.2.4.1 grobcov

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: grobcov(ideal F[,options]);
F: ideal in Q[a][x] (a=parameters, x=variables) to be
discussed.This is the fundamental routine of the
library. It computes the Groebner Cover of a parametric ideal F in Q[a][x]. See
A. Montes , M. Wibmer, "Groebner Bases for Polynomial
Systems with parameters".
JSC 45 (2010) 1391-1425.)
or the not yet published book
A. Montes. " The Groebner Cover" (Discussing
Parametric Polynomial Systems).
The Groebner Cover of a parametric ideal F consist
of a set of pairs(S_i,B_i), where the S_i are disjoint locally closed segments of the parameter space,
and the B_i are the reducedGroebner bases of the
ideal on every point of S_i. The ideal F must be
defined on a parametric ring Q[a][x] (a=parameters,
x=variables).

Return: The list [[lpp_1,basis_1,segment_1], ...,
[lpp_s,basis_s,segment_s]]
optionally [[lpp_1,basis_1,segment_1,lpph_1], ..., [lpp_s,basis_s,segment_s,lpph_s]]
The lpp are constant over a segment and
correspond to the set of lpp of the reduced
Groebner basis for each point of the segment.
With option ("showhom",1) the lpph will be
shown: The lpph corresponds to the lpp of the
homogenized ideal and is different for each
segment. It is given as a string, and shown
only for information. With the default option
"can",1, the segments have different lpph.
Basis: to each element of lpp corresponds
an I-regular function given in full
representation (by option ("ext",1)) or
in generic representation (default option ("ext",0)). The I-regular function is the corresponding

element of the reduced Groebner basis for
each point of the segment with the given lpp.
For each point in the segment, the polynomial
or the set of polynomials representing it,
if they do not specialize to 0, then after
normalization, specializes to the corresponding
element of the reduced Groebner basis.
In the full representation at least one of the
polynomials representing the I-regular function
specializes to non-zero.
With the default option ("rep",0) the
representation of the segment is the
P-representation.
With option ("rep",1) the representation
of the segment is the C-representation.
With option ("rep",2) both representations
of the segment are given.
The P-representation of a segment is of the form
[[p_1,[p_11,..,p_1k1]],..,[p_r,[p_r1,..,p_rkr]]]
representing the segment
Union_i (V(p_i) \ (Union_j V(p_ij))),
where the p's are prime ideals.
The C-representation of a segment is of the form
(E,N) representing V(E) \ V(N), and the ideals E
and N are radical and N contains E.

Options: An option is a pair of arguments: string,
integer. To modify the default options, pairs
of arguments -option name, value- of valid options
must be added to the call.
"null",ideal E: The default is ("null",ideal(0)).
"nonnull",ideal N: The default is
("nonnull",ideal(1)).
When options "null" and/or "nonnull" are given,
then the parameter space is restricted to V(E) \ V(N). "can",0-1: The default is ("can",1).
With the default option the homogenized
ideal is computed before obtaining the Groebner
Cover, so that the result is the canonical Groebner
Cover. Setting ("can",0) only homogenizes the
basis so the result is not exactly canonical,
but the computation is shorter.
"ext",0-1: The default is ("ext",0).
With the default ("ext",0), only the generic
representation of the bases is computed
(single polynomials, but not specializing
to non-zero for every point of the segment.
With option ("ext",1) the full representation
of the bases is computed (possible sheaves)
and sometimes a simpler result is obtained,
but the computation is more time consuming.

"rep",0-1-2: The default is ("rep",0)
and then the segments are given in canonical
P-representation.
Option ("rep",1) represents the segments
in canonical C-representation, and
option ("rep",2) gives both representations.
"comment",0-3: The default is ("comment",0).
Setting "comment" higher will provide
information about the development of the
computation.
"showhom",0-1: The default is ("showhom",0).
Setting "showhom",1 will output the set
of lpp of the homogenized ideal of each segment
as last element. One can give none or whatever
of these options.

Note: The basering R, must be of the form Q[a][x],
(a=parameters, x=variables), and
should be defined previously. The ideal
must be defined on R.

Example:

```
LIB "grobcov.lib";
// EXAMPLE 1:
// Casas conjecture for degree 4:
// Casas-Alvero conjecture states that on a field of characteristic 0,
// if a polynomial of degree n in x has a common root whith each of its
// n-1 derivatives (not assumed to be the same), then it is of the form
// P(x) = k(x + a)^n, i.e. the common roots must all be the same.
if(defined(R)){kill R;}
ring R=(0,a0,a1,a2,a3,a4),(x1,x2,x3),dp;
short=0;
ideal F=x1^4+(4*a3)*x1^3+(6*a2)*x1^2+(4*a1)*x1+(a0),
x1^3+(3*a3)*x1^2+(3*a2)*x1+(a1),
x2^4+(4*a3)*x2^3+(6*a2)*x2^2+(4*a1)*x2+(a0),
x2^2+(2*a3)*x2+(a2),
x3^4+(4*a3)*x3^3+(6*a2)*x3^2+(4*a1)*x3+(a0),
x3+(a3);
grobcov(F);
↦ [1]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             [1]:
↦                _[1]=(a2-a3^2)
↦                _[2]=(a1-a3^3)
↦                _[3]=(a0-a3^4)
↦ [2]:
↦    [1]:
↦       _[1]=x3
↦       _[2]=x2^2
↦       _[3]=x1^3
↦    [2]:
↦       _[1]=x3+(a3)
↦       _[2]=x2^2+(2*a3)*x2+(a3^2)
↦       _[3]=x1^3+(3*a3)*x1^2+(3*a3^2)*x1+(a3^3)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a2-a3^2)
↦             _[2]=(a1-a3^3)
↦             _[3]=(a0-a3^4)
↦          [2]:
↦             [1]:
↦                _[1]=1
// EXAMPLE 2
// M. Rychlik robot;
// Complexity and Applications of Parametric Algorithms of
// Computational Algebraic Geometry.;
// In: Dynamics of Algorithms, R. de la Llave, L. Petzold and J. Lorenz eds.;
// IMA Volumes in Mathematics and its Applications,
// Springer-Verlag 118: 1-29 (2000).;
// (18. Mathematical robotics: Problem 4, two-arm robot).
if (defined(R)){kill R;}
ring R=(0,a,b,l2,l3),(c3,s3,c1,s1), dp;
short=0;
ideal S12=a-l3*c3-l2*c1,b-l3*s3-l2*s1,c1^2+s1^2-1,c3^2+s3^2-1;
S12;
↦ S12[1]=(-l3)*c3+(-l2)*c1+(a)
↦ S12[2]=(-l3)*s3+(-l2)*s1+(b)
↦ S12[3]=c1^2+s1^2-1
↦ S12[4]=c3^2+s3^2-1
grobcov(S12);
↦ [1]:
↦    [1]:
↦       _[1]=c1
↦       _[2]=s3
↦       _[3]=c3
↦       _[4]=s1^2
↦    [2]:
↦       _[1]=(2*a*l2)*c1+(2*b*l2)*s1+(-a^2-b^2-l2^2+l3^2)
↦       _[2]=(l3)*s3+(l2)*s1+(-b)
↦       _[3]=(2*a*l3)*c3+(-2*b*l2)*s1+(-a^2+b^2+l2^2-l3^2)
↦       _[4]=(4*a^2*l2^2+4*b^2*l2^2)*s1^2+(-4*a^2*b*l2-4*b^3*l2-4*b*l2^3+4*\
   b*l2*l3^2)*s1+(a^4+2*a^2*b^2-2*a^2*l2^2-2*a^2*l3^2+b^4+2*b^2*l2^2-2*b^2*l\
   3^2+l2^4-2*l2^2*l3^2+l3^4)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦             [2]:
↦                _[1]=(l2)
↦             [3]:
↦                _[1]=(a^2+b^2)
↦             [4]:
↦                _[1]=(a)
↦ [2]:
↦    [1]:
↦       _[1]=s1
↦       _[2]=s3
↦       _[3]=c3
↦       _[4]=c1^2
↦    [2]:
↦       _[1]=(2*b*l2)*s1+(-b^2-l2^2+l3^2)
↦       _[2]=(2*b*l3)*s3+(-b^2+l2^2-l3^2)
↦       _[3]=(l3)*c3+(l2)*c1
↦       _[4]=(4*b^2*l2^2)*c1^2+(b^4-2*b^2*l2^2-2*b^2*l3^2+l2^4-2*l2^2*l3^2+\
  l3^4)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(a)
↦             [2]:
↦                _[1]=(l2)
↦                _[2]=(a)
↦             [3]:
↦                _[1]=(b)
↦                _[2]=(a)
↦ [3]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(b)
↦             _[2]=(a)
↦          [2]:
↦             [1]:
↦                _[1]=(l2+l3)
↦                _[2]=(b)
↦                _[3]=(a)
↦             [2]:
↦                _[1]=(l3)
↦                _[2]=(b)
↦                _[3]=(a)
↦             [3]:
↦                _[1]=(l2-l3)
↦                _[2]=(b)
↦                _[3]=(a)
↦       [2]:
↦          [1]:
↦             _[1]=(l2)
↦          [2]:
↦             [1]:
↦                _[1]=(l2)
↦                _[2]=(a^2+b^2-l3^2)
↦             [2]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦ [4]:
↦    [1]:
↦       _[1]=s3
↦       _[2]=c3
↦       _[3]=c1^2
↦    [2]:
↦       _[1]=(l2^2*l3+2*l2^2-l3^3)*s3+(2*l2*l3)*s1+(b*l3^2)
↦       _[2]=(l2^2*l3+2*l2^2-l3^3)*c3+(2*l2*l3)*c1+(a*l3^2)
↦       _[3]=c1^2+s1^2-1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l2-l3)
↦             _[2]=(b)
↦             _[3]=(a)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(b)
↦                _[4]=(a)
↦       [2]:
↦          [1]:
↦             _[1]=(l2+l3)
↦             _[2]=(b)
↦             _[3]=(a)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(b)
↦                _[4]=(a)
↦       [3]:
↦          [1]:
↦             _[1]=(l2)
↦             _[2]=(a^2+b^2-l3^2)
↦          [2]:
↦             [1]:
```

```
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(a^2+b^2)
↦ [5]:
↦    [1]:
↦       _[1]=c1^2
↦       _[2]=c3^2
↦    [2]:
↦       _[1]=c1^2+s1^2-1
↦       _[2]=c3^2+s3^2-1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l3)
↦             _[2]=(l2)
↦             _[3]=(b)
↦             _[4]=(a)
↦          [2]:
↦             [1]:
↦                _[1]=1
↦ [6]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l3)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(a^2+b^2-l2^2)
↦             [2]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦ [7]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l3)
↦             _[2]=(l2)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(a^2+b^2)
↦ [8]:
↦    [1]:
```

```
↦       _[1]=s1
↦       _[2]=c1
↦       _[3]=c3^2
↦    [2]:
↦       _[1]=(l2)*s1+(-b)
↦       _[2]=(l2)*c1+(-a)
↦       _[3]=c3^2+s3^2-1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l3)
↦             _[2]=(a^2+b^2-l2^2)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(a^2+b^2)
↦ [9]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l3)
↦             _[2]=(l2)
↦             _[3]=(a^2+b^2)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(b)
↦                _[4]=(a)
↦ [10]:
↦    [1]:
↦       _[1]=s1
↦       _[2]=c1
↦       _[3]=s3
↦       _[4]=c3
↦    [2]:
↦       _[1]=(4*b*l2^3-4*b*l2*l3^2)*s1+(-4*b^2*l2^2-l2^4+2*l2^2*l3^2-l3^4)
↦       _[2]=(4*b^2*l2^3-4*b^2*l2*l3^2)*c1+(-4*a*b^2*l2^2+a*l2^4-2*a*l2^2*l\
  3^2+a*l3^4)
↦       _[3]=(4*b*l2^2*l3-4*b*l3^3)*s3+(4*b^2*l3^2+l2^4-2*l2^2*l3^2+l3^4)
↦       _[4]=(4*b^2*l2^2*l3-4*b^2*l3^3)*c3+(4*a*b^2*l3^2-a*l2^4+2*a*l2^2*l3\
  ^2-a*l3^4)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a^2+b^2)
↦          [2]:
↦             [1]:
```

```
↦                _[1]=(l2+l3)
↦                _[2]=(a^2+b^2)
↦             [2]:
↦                _[1]=(l3)
↦                _[2]=(a^2+b^2)
↦             [3]:
↦                _[1]=(l2-l3)
↦                _[2]=(a^2+b^2)
↦             [4]:
↦                _[1]=(l2)
↦                _[2]=(a^2+b^2)
↦             [5]:
↦                _[1]=(b)
↦                _[2]=(a)
↦ [11]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(l2-l3)
↦             _[2]=(a^2+b^2)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(a^2+b^2)
↦             [2]:
↦                _[1]=(l2-l3)
↦                _[2]=(b)
↦                _[3]=(a)
↦       [2]:
↦          [1]:
↦             _[1]=(l2+l3)
↦             _[2]=(a^2+b^2)
↦          [2]:
↦             [1]:
↦                _[1]=(l3)
↦                _[2]=(l2)
↦                _[3]=(a^2+b^2)
↦             [2]:
↦                _[1]=(l2+l3)
↦                _[2]=(b)
↦                _[3]=(a)
```

D.2.4.2 cgsdr

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: cgsdr(ideal F);
F: ideal in Q[a][x] (a=parameters, x=variables) to be
discussed. Computes a disjoint, reduced Comprehensive

Groebner System (CGS). cgsdr is the starting point of
the fundamental routine grobcov.
The basering R, must be of the form Q[a][x],
(a=parameters, x=variables),
and should be defined previously.

Return: Returns a list T describing a reduced and disjoint
Comprehensive Groebner System (CGS). The output
is a list of (full,hole,basis), where the ideals
full and hole represent the segment V(full) \ V(hole). With option ("out",0) the segments are grouped
by leading power products (lpp) of the reduced
Groebner basis and given in P-representation.
The returned list is of the form:
[[lpp, [num,basis,segment],...,
[num,basis,segment],lpph], ... ,
[lpp, [num,basis,segment],...,
[num,basis,segment],lpph]].
The bases are the reduced Groebner bases (after
normalization) for each point of the corresponding
segment. The third element lpph of each lpp
segment is the lpp of the homogenized ideal
used ideal in the CGS as a string, that
is shown only when option ("can",1) is used.
With option ("can",0) the homogenized basis is used.
With option ("can",1) the homogenized ideal is used.
With option ("can",2) the given basis is used.
With option ("out",1) (default) only KSW is applied and segments are given as difference of varieties and are not grouped The returned list is of the form:
[[E,N,B],..[E,N,B]]
E is the top variety
N is the hole variety.
Segment = V(E) \ V(N)
B is the reduced Groebner basis

Options: An option is a pair of arguments: string, integer.
To modify the default options, pairs of arguments
-option name, value- of valid options must be
added to the call. Inside grobcov the default option is "can",1. It can be used also with option
"can",0 but then the output is not the canonical
Groebner Cover. grobcov cannot be used with
option "can",2.
When cgsdr is called directly, the options are
"can",0-1-2: The default value is "can",2.
In this case no homogenization is done. With option
("can",0) the given basis is homogenized,
and with option ("can",1) the whole given ideal
is homogenized before computing the cgs
and dehomogenized after.
With option ("can",0) the homogenized basis is used. With option ("can",1) the ho-

mogenized ideal is used. With option ("can",2) the given basis is used.
"null",ideal E: The default is ("null",ideal(0)).
"nonnull",ideal N: The default ("nonnull",ideal(1)). When options "null" and/or "nonnull" are given,
then the parameter space is restricted to V(E) \ V(N). "comment",0-1: The default is ("comment",0).
Setting ("comment",1) will provide information
about the development of the computation.
"out",0-1: (default is 1) the output segments are
given as as difference of varieties.
With option "out",0 the output segments are
given in P-representation and the segments
grouped by lpp.
With options ("can",0) and ("can",1) the option
("out",1) is set to ("out",0) because
it is not compatible.
One can give none or whatever of these options.
With the default options ("can",2,"out",1),
only the Kapur-Sun-Wang algorithm is computed.
The algorithm used is:
D. Kapur, Y. Sun, and D.K. Wang "A New Algorithm
for Computing Comprehensive Groebner Systems".
Proceedings of ISSAC'2010, ACM Press, (2010), 29-36. It is very efficient but is only the starting point
for the computation of grobcov.
When grobcov is computed, the call to cgsdr
inside uses specific options that are
more expensive ("can",0-1,"out",0).

Example:

```
LIB "grobcov.lib";
// EXAMPLE:
// Casas conjecture for degree 4:
// Casas-Alvero conjecture states that on a field of characteristic 0,
// if a polynomial of degree n in x has a common root whith each of its
// n-1 derivatives (not assumed to be the same), then it is of the form
// P(x) = k(x + a)^n, i.e. the common roots must all be the same.
if(defined(R)){kill R;}
ring R=(0,a0,a1,a2,a3,a4),(x1,x2,x3),dp;
short=0;
ideal F=x1^4+(4*a3)*x1^3+(6*a2)*x1^2+(4*a1)*x1+(a0),
x1^3+(3*a3)*x1^2+(3*a2)*x1+(a1),
x2^4+(4*a3)*x2^3+(6*a2)*x2^2+(4*a1)*x2+(a0),
x2^2+(2*a3)*x2+(a2),
x3^4+(4*a3)*x3^3+(6*a2)*x3^2+(4*a1)*x3+(a0),
x3+(a3);
cgsdr(F);
↦ [1]:
↦    [1]:
↦       _[1]=0
↦    [2]:
↦       _[1]=(-a2+a3^2)
↦          _[2]=(-a1*a2+a1*a3^2+3*a2^2*a3-5*a2*a3^3+2*a3^5)
↦          _[3]=(23*a1^2-138*a1*a2*a3+92*a1*a3^3+25*a2^3+132*a2^2*a3^2-201*a2*\
   a3^4+67*a3^6)
↦          _[4]=(a0-4*a1*a3+6*a2*a3^2-3*a3^4)
↦       [3]:
↦          _[1]=1
↦    [2]:
↦       [1]:
↦          _[1]=(a2-a3^2)
↦          _[2]=(a1-a3^3)
↦          _[3]=(a0-a3^4)
↦       [2]:
↦          _[1]=1
↦       [3]:
↦          _[1]=x3+(a3)
↦          _[2]=x2^2+(2*a3)*x2+(a3^2)
↦          _[3]=x1^3+(3*a3)*x1^2+(3*a3^2)*x1+(a3^3)
```

D.2.4.3 pdivi

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: pdivi(poly f,ideal F);
poly f: the polynomial in Q[a][x] to be divided
ideal F: the divisor ideal in Q[a][x].
(a=parameters, x=variables).

Return: A list (poly r, ideal q, poly m). r is the remainder of the pseudodivision, q is the set of quotients, and m is the coefficient factor by which f is to be multiplied.

Note: pseudodivision of a poly f by an ideal F in Q[a][x].
Returns a list (r,q,m) such that
m*f=r+sum(q.F),
and no lpp of a divisor divides a pp of r.

Example:

```
LIB "grobcov.lib";
"RXAMPLE:";
↦ RXAMPLE:
// Division of a polynom by an ideal
if(defined(R)){kill R;}
ring R=(0,a,b,c),(x,y),dp;
short=0;
// Divisor
poly f=(ab-ac)*xy+(ab)*x+(5c);
// Dividends
ideal F=ax+b,
cy+a;
// (Remainder, quotients, factor)
def r=pdivi(f,F);
r;
↦ [1]:
```

```
↦    (a*b^2-a*b*c-b^2*c+5*c^2)
↦ [2]:
↦    _[1]=(b*c-c^2)*y+(b*c)
↦    _[2]=(-b^2+b*c)
↦ [3]:
↦    (c)
// Verifying the division
r[3]*f-(r[2][1]*F[1]+r[2][2]*F[2]+r[1]);
↦ 0
```

D.2.4.4 pnormalf

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: pnormalf(poly f,ideal E,ideal N);
f: the polynomial in Q[a][x] (a=parameters,
x=variables) to be reduced modulo V(E) \ V(N)
of a segment in Q[a].
E: the null conditions ideal in Q[a]
N: the non-null conditions in Q[a]

Return: a reduced polynomial g of f, whose coefficients are
reduced modulo E and having no factor in N.

Note: Should be called from ring Q[a][x]. Ideals E and N must be given by polynomials in Q[a].

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,a,b,c),(x,y),dp;
short=0;
// parametric polynom
poly f=(b^2-1)*x^3*y+(c^2-1)*x*y^2+(c^2*b-b)*x+(a-bc)*y;
// ideals defining V(p)\V(q)
ideal p=c-1;
ideal q=a-b;
// pnormaform of f on V(p) \ V(q)
pnormalf(f,p,q);
↦ (b^2-1)*x^3*y+(a-b)*y
```

D.2.4.5 Crep

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: Crep(ideal N,ideal M);
ideal N (null ideal) (not necessarily radical nor
maximal) in Q[a]. (a=parameters, x=variables).
ideal M (hole ideal) (not necessarily containing N)
in Q[a]. To be called in a ring Q[a][x] or a ring Q[a]. But the ideals can contain only the parameters
in Q[a].

Return: The canonical C-representation [P,Q] of the
locally closed set, formed by a pair of radical

ideals with P included in Q, representing the set
V(P) \ V(Q) = V(N) \ V(M)

Example:

```
LIB "grobcov.lib";
short=0;
if(defined(R)){kill R;}
ring R=0,(a,b,c),lp;
ideal p=a*b;
ideal q=a,b-2;
// C-representation of V(p) \ V(q)
Crep(p,q);
↦ [1]:
↦    _[1]=ab
↦ [2]:
↦    _[1]=b-2
↦    _[2]=a
```

D.2.4.6 Prep

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: Prep(ideal N,ideal M);
ideal N (null ideal) (not necessarily radical nor maximal) in Q[a]. (a=parameters, x=variables).
ideal M (hole ideal) (not necessarily containing N) in Q[a]. To be called in a ring Q[a][x] or a ring Q[a]. But the ideals can contain only the parameters in Q[a].

Return: The canonical P-representation of the locally closed set V(N) \ V(M)
Output: [Comp_1, .. , Comp_s] where
Comp_i=[p_i,[p_i1,..,p_is_i]]

Example:

```
LIB "grobcov.lib";
short=0;
if(defined(R)){kill R;}
ring R=0,(a,b,c),lp;
ideal p=a*b;;
ideal q=a,b-1;
// P-representation of V(p) \ V(q)
Prep(p,q);
↦ [1]:
↦    [1]:
↦       _[1]=b
↦    [2]:
↦       [1]:
↦          _[1]=1
↦ [2]:
↦    [1]:
↦       _[1]=a
↦    [2]:
```

```
↦         [1]:
↦            _[1]=b-1
↦            _[2]=a
```

D.2.4.7 PtoCrep

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: PtoCrep(list L)
list L= [Comp_1, .. , Comp_s] where
list Comp_i=[p_i,[p_i1,..,p_is_i]], is the
P-representation of a locally closed set
V(N) \ V(M). To be called in a ring Q[a][x]
or a ring Q[a]. But the ideals can contain
only the parameters in Q[a].

Return: The canonical C-representation [P,Q] of the
locally closed set. A pair of radical ideals with
P included in Q, representing the
set V(P) \ V(Q)

Example:

```
LIB "grobcov.lib";
//EXAMPLE:
if(defined(R)){kill R;}
ring R=0,(a,b,c),lp;
short=0;
ideal p=a*(a^2+b^2+c^2-25);
ideal q=a*(a-3),b-4;
// C-representaion of V(p) \ V(q)
def Cr=Crep(p,q);
Cr;
↦ [1]:
↦    _[1]=a^3+a*b^2+a*c^2-25*a
↦ [2]:
↦    _[1]=b-4
↦    _[2]=a*c
↦    _[3]=a^2-3*a
// P-representation of V(p) \ V(q)
def L=Prep(p,q);
L;
↦ [1]:
↦    [1]:
↦       _[1]=a^2+b^2+c^2-25
↦    [2]:
↦       [1]:
↦          _[1]=c-3
↦          _[2]=b-4
↦          _[3]=a
↦       [2]:
↦          _[1]=c+3
↦          _[2]=b-4
↦          _[3]=a
↦       [3]:
```

```
↦             _[1]=c
↦             _[2]=b-4
↦             _[3]=a-3
↦  [2]:
↦     [1]:
↦        _[1]=a
↦     [2]:
↦        [1]:
↦           _[1]=b-4
↦           _[2]=a
PtoCrep(L);
↦  [1]:
↦     _[1]=a^3+a*b^2+a*c^2-25*a
↦  [2]:
↦     _[1]=b-4
↦     _[2]=a*c
↦     _[3]=a^2-3*a
```

D.2.4.8 extendpoly

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: extendGC(poly f,ideal p,ideal q);
f is a polynomial in Q[a][x] in generic representation of an I-regular function F defined on the locally
closed segment S=V(p) \ V(q).
p,q are ideals in Q[a], representing the Crep of
segment S.

Return: the extended representation of F in S.
It can consist of a single polynomial or a set of
polynomials when needed.

Note: The basering R, must be of the form Q[a][x],
(a=parameters,x=variables), and should be
defined previously. The ideals must be defined on R.

Example:

```
LIB "grobcov.lib";
// EXAMPLE 1
if(defined(R)){kill R;}
ring R=(0,a1,a2),(x),lp;
short=0;
poly f=(a1^2-4*a1+a2^2-a2)*x+(a1^4-16*a1+a2^3-4*a2);
ideal p=a1*a2;
ideal q=a2^2-a2,a1*a2,a1^2-4*a1;
extendpoly(f,p,q);
↦  (a1+4*a2-4)*x+(a1^3+4*a2^2-16)
// EXAMPLE 2
if (defined(R)){kill R;}
ring R=(0,a0,b0,c0,a1,b1,c1,a2,b2,c2),(x), dp;
short=0;
poly f=(b1*a2*c2-c1*a2*b2)*x+(-a1*c2^2+b1*b2*c2+c1*a2*c2-c1*b2^2);
ideal p=
(-a0*b1*c2+a0*c1*b2+b0*a1*c2-b0*c1*a2-c0*a1*b2+c0*b1*a2),
(a1^2*c2^2-a1*b1*b2*c2-2*a1*c1*a2*c2+a1*c1*b2^2+b1^2*a2*c2-b1*c1*a2*b2+c1^2*a2^2),
(a0*a1*c2^2-a0*b1*b2*c2-a0*c1*a2*c2+a0*c1*b2^2+b0*b1*a2*c2-b0*c1*a2*b2
- c0*a1*a2*c2+c0*c1*a2^2),
(a0^2*c2^2-a0*b0*b2*c2-2*a0*c0*a2*c2+a0*c0*b2^2+b0^2*a2*c2-b0*c0*a2*b2+c0^2*a2^2),
(a0*a1*c1*c2-a0*b1^2*c2+a0*b1*c1*b2-a0*c1^2*a2+b0*a1*b1*c2-b0*a1*c1*b2
-c0*a1^2*c2+c0*a1*c1*a2),
(a0^2*c1*c2-a0*b0*b1*c2-a0*c0*a1*c2+a0*c0*b1*b2-a0*c0*c1*a2+b0^2*a1*c2
-b0*c0*a1*b2+c0^2*a1*a2),
(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*c1-b0*c0*a1*b1+c0^2*a1^2),
(2*a0*a1*b1*c1*c2-a0*a1*c1^2*b2-a0*b1^3*c2+a0*b1^2*c1*b2-a0*b1*c1^2*a2
-b0*a1^2*c1*c2+b0*a1*b1^2*c2-b0*a1*b1*c1*b2+b0*a1*c1^2*a2-c0*a1^2*b1*c2+c0*a1^2*c1*b2
ideal q=
(-a1*c2+c1*a2),
(-a1*b2+b1*a2),
(-a0*c2+c0*a2),
(-a0*b2+b0*a2),
(-a0*c1+c0*a1),
(-a0*b1+b0*a1),
(-a1*b1*c2+a1*c1*b2),
(-a0*b1*c2+a0*c1*b2),
(-a0*b0*c2+a0*c0*b2),
(-a0*b0*c1+a0*c0*b1);
extendpoly(f,p,q);
↦ _[1]=(-a1*b2+b1*a2)*x+(-a1*c2+c1*a2)
↦ _[2]=(-a0*b2+b0*a2)*x+(-a0*c2+c0*a2)
↦ _[3]=(-a0*b1+b0*a1)*x+(-a0*c1+c0*a1)
```

D.2.4.9 extendGC

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: extendGC(list GC);
list GC must the grobcov of a parametric ideal computed with option "rep",2. It determines the full
representation.
The default option of grobcov provides the bases in
generic representation (the I-regular functions forming the bases are then given by a single polynomial.
They can specialize to zero for some points of the
segments, but in general, it is sufficient for many
purposes. Nevertheless the I-regular functions allow a full representation given by a set of polynomials
specializing to the value of the function (after
normalization) or to zero, but at least one of the
polynomials specializes to non-zero. The full
representation can be obtained by computing
the grobcov with option "ext",1. (The default
option there is "ext",0).
With option "ext",1 the computation can be
much more time consuming, but the result can
be simpler.
Alternatively, one can compute the full representation of the bases after computing

grobcov with the default
option for "ext" and the option "rep",2,
that outputs both the Prep and the Crep of the
segments, and then call "extendGC" to its output.

Return: When calling extendGC(grobcov(S,"rep",2)) the
result is of the form
[[[lpp_1,basis_1,segment_1,lpph_1], ... ,
[lpp_s,basis_s,segment_s,lpph_s]]],
where each function of the basis can be given
by an ideal of representants.

Note: The basering R, must be of the form Q[a][x],
(a=parameters, x=variables),
and should be defined previously. The ideal
must be defined on R.

discussion of parametric ideal

Example:

```
LIB "grobcov.lib";
// EXAMPLE
if(defined(R)){kill R;}
ring R=(0,a0,b0,c0,a1,b1,c1),(x), dp;
short=0;
ideal S=a0*x^2+b0*x+c0,
a1*x^2+b1*x+c1;
def GCS=grobcov(S,"rep",2);
// grobcov(S) with both P and C representations
GCS;
↦ [1]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             [1]:
↦                _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*\
   a1*c1-b0*c0*a1*b1+c0^2*a1^2)
↦    [4]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*c1-\
   b0*c0*a1*b1+c0^2*a1^2)
↦ [2]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=(b0*a1*c1-c0*a1*b1)*x+(-a0*c1^2+b0*b1*c1+c0*a1*c1-c0*b1^2)
↦    [3]:
```

```
↦       [1]:
↦          [1]:
↦             _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*\
  c1-b0*c0*a1*b1+c0^2*a1^2)
↦          [2]:
↦             [1]:
↦                _[1]=(b0*c1-c0*b1)
↦                _[2]=(a0*c1-c0*a1)
↦                _[3]=(a0*b1-b0*a1)
↦             [2]:
↦                _[1]=(a1)
↦                _[2]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*c1-\
  b0*c0*a1*b1+c0^2*a1^2)
↦       [2]:
↦          _[1]=(-a0*c1+c0*a1)
↦          _[2]=(-a0*b1+b0*a1)
↦          _[3]=(-a0*b0*c1+a0*c0*b1)
↦ [3]:
↦    [1]:
↦       _[1]=x^2
↦    [2]:
↦       _[1]=(a1)*x^2+(b1)*x+(c1)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(b0*c1-c0*b1)
↦             _[2]=(a0*c1-c0*a1)
↦             _[3]=(a0*b1-b0*a1)
↦          [2]:
↦             [1]:
↦                _[1]=(a1)
↦                _[2]=(b0*c1-c0*b1)
↦                _[3]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(-b0*c1+c0*b1)
↦          _[2]=(-a0*c1+c0*a1)
↦          _[3]=(-a0*b1+b0*a1)
↦       [2]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦          _[3]=(-b0*c1+c0*b1)
↦ [4]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
```

```
↦             _[1]=(b1)
↦             _[2]=(a1)
↦             _[3]=(b0)
↦             _[4]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=(c1)
↦                _[2]=(b1)
↦                _[3]=(a1)
↦                _[4]=(c0)
↦                _[5]=(b0)
↦                _[6]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(b1)
↦          _[2]=(a1)
↦          _[3]=(b0)
↦          _[4]=(a0)
↦       [2]:
↦          _[1]=(c1)
↦          _[2]=(b1)
↦          _[3]=(a1)
↦          _[4]=(c0)
↦          _[5]=(b0)
↦          _[6]=(a0)
↦ [5]:
↦    [1]:
↦       _[1]=0
↦    [2]:
↦       _[1]=0
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(c1)
↦             _[2]=(b1)
↦             _[3]=(a1)
↦             _[4]=(c0)
↦             _[5]=(b0)
↦             _[6]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=1
↦    [4]:
↦       [1]:
↦          _[1]=(c1)
↦          _[2]=(b1)
↦          _[3]=(a1)
↦          _[4]=(c0)
↦          _[5]=(b0)
↦          _[6]=(a0)
↦       [2]:
↦          _[1]=1
↦ [6]:
```

```
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=(b1)*x+(c1)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a1)
↦             _[2]=(b0*c1-c0*b1)
↦             _[3]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=(b1)
↦                _[2]=(a1)
↦                _[3]=(b0)
↦                _[4]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦          _[3]=(-b0*c1+c0*b1)
↦       [2]:
↦          _[1]=(b1)
↦          _[2]=(a1)
↦          _[3]=(b0)
↦          _[4]=(a0)
↦ [7]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a1)
↦             _[2]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=(a1)
↦                _[2]=(b0*c1-c0*b1)
↦                _[3]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦       [2]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦          _[3]=(-b0*c1+c0*b1)
def FGC=extendGC(GCS,"rep",1);
// Full representation
FGC;
↦ [1]:
```

```
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             [1]:
↦                _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*\
  a1*c1-b0*c0*a1*b1+c0^2*a1^2)
↦    [4]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*c1-\
  b0*c0*a1*b1+c0^2*a1^2)
↦ [2]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=(a0*b1-b0*a1)*x+(a0*c1-c0*a1)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*\
  c1-b0*c0*a1*b1+c0^2*a1^2)
↦          [2]:
↦             [1]:
↦                _[1]=(b0*c1-c0*b1)
↦                _[2]=(a0*c1-c0*a1)
↦                _[3]=(a0*b1-b0*a1)
↦             [2]:
↦                _[1]=(a1)
↦                _[2]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(a0^2*c1^2-a0*b0*b1*c1-2*a0*c0*a1*c1+a0*c0*b1^2+b0^2*a1*c1-\
  b0*c0*a1*b1+c0^2*a1^2)
↦       [2]:
↦          _[1]=(-a0*c1+c0*a1)
↦          _[2]=(-a0*b1+b0*a1)
↦          _[3]=(-a0*b0*c1+a0*c0*b1)
↦ [3]:
↦    [1]:
↦       _[1]=x^2
↦    [2]:
↦       [1]:
↦          _[1]=(a1)*x^2+(b1)*x+(c1)
↦          _[2]=(a0*a1)*x^2+(b0*a1)*x+(c0*a1)
↦    [3]:
↦       [1]:
```

```
↦          [1]:
↦             _[1]=(b0*c1-c0*b1)
↦             _[2]=(a0*c1-c0*a1)
↦             _[3]=(a0*b1-b0*a1)
↦          [2]:
↦             [1]:
↦                _[1]=(a1)
↦                _[2]=(b0*c1-c0*b1)
↦                _[3]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(-b0*c1+c0*b1)
↦          _[2]=(-a0*c1+c0*a1)
↦          _[3]=(-a0*b1+b0*a1)
↦       [2]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦          _[3]=(-b0*c1+c0*b1)
↦ [4]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(b1)
↦             _[2]=(a1)
↦             _[3]=(b0)
↦             _[4]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=(c1)
↦                _[2]=(b1)
↦                _[3]=(a1)
↦                _[4]=(c0)
↦                _[5]=(b0)
↦                _[6]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(b1)
↦          _[2]=(a1)
↦          _[3]=(b0)
↦          _[4]=(a0)
↦       [2]:
↦          _[1]=(c1)
↦          _[2]=(b1)
↦          _[3]=(a1)
↦          _[4]=(c0)
↦          _[5]=(b0)
↦          _[6]=(a0)
↦ [5]:
↦    [1]:
```

```
↦       _[1]=0
↦    [2]:
↦       _[1]=0
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(c1)
↦             _[2]=(b1)
↦             _[3]=(a1)
↦             _[4]=(c0)
↦             _[5]=(b0)
↦             _[6]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=1
↦    [4]:
↦       [1]:
↦          _[1]=(c1)
↦          _[2]=(b1)
↦          _[3]=(a1)
↦          _[4]=(c0)
↦          _[5]=(b0)
↦          _[6]=(a0)
↦       [2]:
↦          _[1]=1
↦ [6]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       [1]:
↦          _[1]=(b1)*x+(c1)
↦          _[2]=(b0)*x+(c0)
↦    [3]:
↦       [1]:
↦          [1]:
↦             _[1]=(a1)
↦             _[2]=(b0*c1-c0*b1)
↦             _[3]=(a0)
↦          [2]:
↦             [1]:
↦                _[1]=(b1)
↦                _[2]=(a1)
↦                _[3]=(b0)
↦                _[4]=(a0)
↦    [4]:
↦       [1]:
↦          _[1]=(a1)
↦          _[2]=(a0)
↦          _[3]=(-b0*c1+c0*b1)
↦       [2]:
↦          _[1]=(b1)
↦          _[2]=(a1)
↦          _[3]=(b0)
↦       _[4]=(a0)
↦ [7]:
↦   [1]:
↦     _[1]=1
↦   [2]:
↦     _[1]=1
↦   [3]:
↦     [1]:
↦       [1]:
↦         _[1]=(a1)
↦         _[2]=(a0)
↦       [2]:
↦         [1]:
↦           _[1]=(a1)
↦           _[2]=(b0*c1-c0*b1)
↦           _[3]=(a0)
↦   [4]:
↦     [1]:
↦       _[1]=(a1)
↦       _[2]=(a0)
↦     [2]:
↦       _[1]=(a1)
↦       _[2]=(a0)
↦       _[3]=(-b0*c1+c0*b1)
```

D.2.4.10 locus

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: locus(ideal F [,options])
Special routine for determining the locus of points of a geometrical construction.

Input: The input ideal must be the ideal of the set equations defining the locus, defined in the ring
ring Q(0,a1,..,ap,x1,..xn)(u1,..um,v1,..vn),lp;
Calling sequence:
locus(F [,options]);
a=fixed parameters,x=tracer variables, u=auxiliary variables, v=mover variables. The parameters a are optative. If they are used, then the option "numpar\=,np must be declared, being np the number of fixed parameters. The tracer variables are x1,..xn, where n is the dimension of the space. By default, the mover variables are the last n variables. Its number can be forced by the user to the last
k variables by adding the option "moverdim",k.
Nevertheless, this option is recommended only
to experiment, and can provide incorrect taxonomies.
The remaining variables are auxiliary.

Options: An option is a pair of arguments: string, integer.
To modify the default options, pairs of arguments
-option name, value- of valid options must be added to the call.The algorithm allows the following options as pair of arguments:

"numpar", np in order to consider the first np parameters of the ring to be fixed parameters of the locus, being the tracer variables the remaining parameters.
To be used for a paramteric locus. (New in release N12).

"moverdim", k to force the mover-variables to be the last k variables. This determines the antiimage and its dimension. By defaulat k is equal to the last n variables,
We can experiment with a different value,
but this can produce an error in the character
"Normal" or "Special" of a locus component.

"grobcov", G, where G is the list of a previous computed grobcov(F). It is to be used when we modify externally the grobcov, for example to obtain the real grobcov.

"comments", c: by default it is 0, but it can be set to 1.

Return: The output is a list of the components:
((p1,(p11,..p1s_1),tax_1), .., (pk,(pk1,..pks_k),tax_k) Elements 1 and 2 of a component represent the
P-canonical form of the component.
The third element tax is:
for normal point components,
tax=(d,taxonomy,anti-image) being
d=dimension of the anti-image on the mover variables, taxonomy="Normal" or "Special" and
anti-image=ideal of the anti-image over the mover
variables.
for non-normal point components,
tax =(d,taxonomy) being
d=dimension of the component and
taxonomy="Accumulation" or "Degenerate".
The components are given in canonical P-representation. The normal locus has two kind of components:
Normal and Special.
Normal component:
- each point in the component has 0-dimensional
anti-image.
- the anti-image in the mover coordinates is equal
to the dimension of the component
Special component:
- each point in the component has 0-dimensional
anti-image.
- the anti-image in the mover coordinates has dimension smaller than the dimension of the component
The non-normal locus has two kind of components:
Accumulation and Degenerate.
Accumulation component:
- each point in the component has anti-image of
dimension greater than 0.
- the component has dimension less than n-1.
Degenerate components:
- each point in the component has anti-image
of dimension greater than 0.
- the component has dimension n-1.
When a normal point component has degree greater than 9, then the taxonomy is not determined, and (n,'normal', 0) is returned as third element of the component. (n is the dimension of the tracer space).

Given a parametric ideal F representing the system F
determining the locus of points (x) which verify certain properties, the call to locus(F)
determines the different classes of locus components, following the taxonomy
defined in the book:
A. Montes. "The Groebner Cover"
A previous paper gives particular definitions
for loci in 2d.
M. Abanades, F. Botana, A. Montes, T. Recio,
"An Algebraic Taxonomy for Locus Computation
in Dynamic Geometry",
Computer-Aided Design 56 (2014) 22-33.

Note: The input must be the locus system.

Example:

```
LIB "grobcov.lib";
// EXAMPLE 1
// Conchoid, Pascal's Limacon.
//  1. Given a circle: x1^2+y1^2-4
//  2. and a mover point M(x1,y1) on it
//  3. Consider the fix point P(0,2) on the circle
//  4. Consider the line l passing through M and P
//  5. The tracer T(x,y) are the points on l at fixed distance 1 to M.
if(defined(R)){kill R;}
ring R=(0,x,y),(x1,y1),dp;
short=0;
// Concoid
ideal S96=x1 ^2+y1 ^2-4,(x-2)*x1 -x*y1 +2*x,(x-x1 )^2+(y-y1 )^2-1;
locus(S96);
↦ [1]:
↦    [1]:
↦       _[1]=(x^4+2*x^3+x^2*y^2-3*x^2-2*x*y^2-8*x*y-6*x+2*y^2+8*y+6)
↦    [2]:
↦       [1]:
↦          _[1]=(y^2+2*y+2)
↦          _[2]=(x-y-2)
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=x1^2+y1^2-4
↦ [2]:
↦    [1]:
↦       _[1]=(x^2+y^2-4*y+3)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          0
↦       [2]:
```

```
↦           Special
↦        [3]:
↦           _[1]=65*y1^5-86*y1^4-180*y1^3+120*y1^2+256*y1-256
↦           _[2]=-2015*y1^4+5916*y1^3+5700*y1^2+9216*x1-14400*y1-9088
// EXAMPLE 2
// Consider two parallel planes z1=-1 and z1=1, and two orthogonal parabolas on them
// Determine the locus generated by the lines that rely the two parabolas
// through the points having parallel tangent vectors.
if(defined(R)){kill R;}
ring R=(0,x,y,z),(x2,y2,z2,z1,y1,x1,lam), lp;
short=0;
ideal L=z1+1,
x1^2-y1,
z2-1,
y2^2-x2,
4*x1*y2-1,
x-x1-lam*(x2-x1),
y-y1-lam*(y2-y1),
z-z1-lam*(z2-z1);
locus(L);  // uses "moverdim",3
↦  [1]:
↦     [1]:
↦        _[1]=(2048*x^3*z+2048*x^3-4096*x^2*y^2+1152*x*y*z^2-1152*x*y-2048*y\
   ^3*z+2048*y^3+27*z^4-54*z^2+27)
↦     [2]:
↦        [1]:
↦           _[1]=(z+1)
↦           _[2]=(y)
↦        [2]:
↦           _[1]=(z-1)
↦           _[2]=(x)
↦     [3]:
↦        [1]:
↦           2
↦        [2]:
↦           Normal
↦        [3]:
↦           _[1]=y1-x1^2
// Observe the choose of the mover variables: the last 3 variables y1,x1,lam
// If we choose x1,y1,z1 instead, the taxonomy becomes "Special" because
// z1=-1 is fix and do not really correspond to the mover variables.
// EXAMPLE 3 of parametric locus:
// Determining the equation of a general ellipse;
// Uncentered elipse;
// Parameters  (a,b,a0,b0,p):
//   a=large semiaxis, b=small semiaxis,
//   (a0,b0) = center of the ellipse,
//   (a1,b1) and (2*a0-a1,2*b0-b1) the focus,
//   p the slope of the line of the a-axis of the ellipse.
// Determine the equation of the ellipse.
// We must use the option "numpar",5 in order to consider
// the first 5 parameters as free parameters for the locus
// Auxiliary variabes:
```

```
//  d1=distance from focus (a1,b1) to the mover point M(x1,y1),
//  d2=distance from focus (a2,b2) to the mover point M(x1,y1),
//  f=focus distance= distance from (a0,b0) to (a1,b1).
// Mover point (x1,y1) = tracer point (x,y).
if(defined(R1)){kill R1;}
ring R1=(0,a,b,a0,b0,p,x,y),(d1,d2,f,a1,b1,x1,y1),lp;
ideal F3=b1-b0-p*(a1-a0),
//b2-b0+p*(a1-a0),
//a1+a2-2*a0,
//b1+b2-2*b0,
f^2-(a1-a0)^2-(b1-b0)^2,
f^2-a^2-b^2,
(x1-a1)^2+(y1-b1)^2-d1^2,
(x1-2*a0+a1)^2+(y1-2*b0+b1)^2-d2^2,
d1+d2-2*a,
x-x1,
y-y1;
def G3=grobcov(F3);
def Loc3=locus(F3,"grobcov",G3,"numpar",5); Loc3;
↦ [1]:
↦    [1]:
↦       _[1]=(a^2*b^2*p^2+a^2*b^2+a^2*a0^2*p^2-2*a^2*a0*b0*p-2*a^2*a0*p^2*x\
   +2*a^2*a0*p*y+a^2*b0^2+2*a^2*b0*p*x-2*a^2*b0*y+a^2*p^2*x^2-2*a^2*p*x*y+a^\
   2*y^2-b^2*a0^2-2*b^2*a0*b0*p+2*b^2*a0*p*y+2*b^2*a0*x-b^2*b0^2*p^2+2*b^2*b\
   0*p^2*y+2*b^2*b0*p*x-b^2*p^2*y^2-2*b^2*p*x*y-b^2*x^2)
↦    [2]:
↦       [1]:
↦          _[1]=(p^2+1)
↦          _[2]=(a0+b0*p-p*y-x)
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=(a^2*p^2-b^2)*x1^2+(-2*a^2*p-2*b^2*p)*x1*y1+(-2*a^2*a0*p^2+\
   2*a^2*b0*p+2*b^2*a0+2*b^2*b0*p)*x1+(a^2-b^2*p^2)*y1^2+(2*a^2*a0*p-2*a^2*b\
   0+2*b^2*a0*p+2*b^2*b0*p^2)*y1+(a^2*b^2*p^2+a^2*b^2+a^2*a0^2*p^2-2*a^2*a0*\
   b0*p+a^2*b0^2-b^2*a0^2-2*b^2*a0*b0*p-b^2*b0^2*p^2)
↦ [2]:
↦    [1]:
↦       _[1]=(p^2+1)
↦       _[2]=(a0+b0*p-p*y-x)
↦       _[3]=(b)
↦       _[4]=(a)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          -1
↦       [2]:
↦          Accumulation
```

```
// General ellipse:
def C=Loc3[1][1][1];
C;
↦ (a^2*b^2*p^2+a^2*b^2+a^2*a0^2*p^2-2*a^2*a0*b0*p-2*a^2*a0*p^2*x+2*a^2*a0*p\
   *y+a^2*b0^2+2*a^2*b0*p*x-2*a^2*b0*y+a^2*p^2*x^2-2*a^2*p*x*y+a^2*y^2-b^2*a\
   0^2-2*b^2*a0*b0*p+2*b^2*a0*p*y+2*b^2*a0*x-b^2*b0^2*p^2+2*b^2*b0*p^2*y+2*b\
   ^2*b0*p*x-b^2*p^2*y^2-2*b^2*p*x*y-b^2*x^2)
// Centered ellipse of semiaxes (a,b):
def C0=subst(C,a0,0,b0,0,p,0);
C0;
↦ (a^2*b^2+a^2*y^2-b^2*x^2)
```

D.2.4.11 locusdg

Procedure from library grobcov.lib (see Section D.2.4 [grobcov_lib], page 817).

Usage: locusdg(list L)
Calling sequence:
locusdg(locus(S)).

Return: The output is the list of the "Relevant" components of the locus in Dynamic Geometry
[C1,..,C:m], where
C_i= [p_i,[p_i1,..p_is_i], "Relevant", level_i]
The "Relevant" components are "Normal" and
"Accumulation" components of the locus. (See help
for locus).

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;};
ring R=(0,a,b),(x,y),dp;
short=0;
// Concoid
ideal S96=x^2+y^2-4,(b-2)*x-a*y+2*a,(a-x)^2+(b-y)^2-1;
def L96=locus(S96);
L96;
↦ [1]:
↦    [1]:
↦       _[1]=(a^4+2*a^2*b^2-9*a^2+b^4-9*b^2+4*b+12)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=x^2+y^2-4
↦ [2]:
↦    [1]:
↦       _[1]=(a^2+b^2-4*b+3)
↦    [2]:
↦       [1]:
```

```
↦           _[1]=1
↦     [3]:
↦        [1]:
↦           0
↦        [2]:
↦           Special
↦        [3]:
↦           _[1]=y^2-3*y+2
↦           _[2]=x*y-x
↦           _[3]=x^2+3*y-6
locusdg(L96);
↦  [1]:
↦     [1]:
↦        _[1]=(a^4+2*a^2*b^2-9*a^2+b^4-9*b^2+4*b+12)
↦     [2]:
↦        [1]:
↦           _[1]=1
↦     [3]:
↦        [1]:
↦           1
↦        [2]:
↦           Relevant
↦        [3]:
↦           _[1]=x^2+y^2-4
```

D.2.4.12 envelop

Procedure from library grobcov.lib (see Section D.2.4 [grobcov_lib], page 817).

Usage: envelop(poly F,ideal C[,options]);
poly F must represent the family of hyper-surfaces for which on want to compute its envelop. ideal C must be
the ideal of restrictions on the variables defining the family, and should contain less polynomials than the
number of variables. (x_1,..,x_n) are the variables of the hyper-surfaces of F, that are considered as
parameters of the parametric ring. (u_1,..,u_m) are
the parameteres of the hyper-surfaces, that are
considered as variables of the parametric ring.
In the actual version, parametric envelope are accepted. To include fixed parameters a1,..ap, to the problem, one must declare them as the first parameters of the ring. if the the number of free parameters is p, the option "numpar",p is required.
Calling sequence:
ring R=(0,a1,..,ap,x_1,..,x_n),(u_1,..,u_m),lp;
poly F=F(a1,..ap,x_1,..,x_n,u_1,..,u_m);
ideal C=g_1(a1,..,ap,u_1,..u_m),..,g_s(a1,..ap,u_1,..u_m); envelop(F,C[,options]); where s<m.
x1,..,xn are the tracer variables.
u_1,..,u_m are the auxiliary variables.
a1,..,ap are the fixed parameters if they exist
If the problem is a parametric envelope, and a's exist, then the option "numpar",p m must be given.

By default the las n variables are the mover variables. See the EXAMPLE of parametric envelop by calling
example envelop,

Return: The output is a list of the components [C_1, .. , C_n] of the locus. Each component is given by
Ci=[pi,[pi1,..pi_s_i],tax] where
pi,[pi1,..pi_s_i] is the canonical P-representation of the component.
Concerning tax: (see help for locus)
For normal-point components is
tax=[d,taxonomy,anti-image], being
d=dimension of the anti-image
taxonomy="Normal" or "Special"
anti-image=values of the mover corresponding
to the component
For non-normal-point components is
tax=[d,taxonomy]
d=dimension of the component
taxonomy="Accumulation" or "Degenerate".

Options: An option is a pair of arguments: string, integer.
To modify the default options,
pairs of arguments -option name, value- of valid options must be added to the call.

The algorithm allows the following options as pair of arguments: "comments", c: by default it is 0, but it can be set to 1. "anti-image", a: by default a=1 and the anti-image is shown also for "Normal" components.
For a=0, it is not shown.
"moverdim", k: by default it is equal to n, the number of x-tracer variables.
"numpar",p when fixed parameters are included

Note: grobcov and locus are called internally.
The basering R, must be of the form Q[a,x][u]
(x=variables, u=auxiliary variables), (a fixed parameters). This routine uses the generalized definition of envelop introduced in the book
A. Montes. "The Groebner Cover" (Discussing Parametric
Polynomial Systems) not yet published.

Example:

```
LIB "grobcov.lib";
"EAXMPLE:";
↦ EAXMPLE:
// EXAMPLE 1
// Steiner Deltoid
// 1. Consider the circle x1^2+y1^2-1=0, and a mover point M(x1,y1) on it.
// 2. Consider the triangle A(0,1), B(-1,0), C(1,0).
// 3. Consider lines passing through M perpendicular to two sides of ABC triangle.
// 4. Determine the envelope of the lines above.
if(defined(R)){kill R;}
ring R=(0,x,y),(x1,y1,x2,y2),lp;
short=0;
ideal C=(x1)^2+(y1)^2-1,
x2+y2-1,
x2-y2-x1+y1;
```

```
matrix M[3][3]=x,y,1,x2,y2,1,x1,0,1;
poly F=det(M);
// The lines of family F are
F;
↦ -x1*y2+(y)*x1+(-y)*x2+(x)*y2
// The conditions C are
C;
↦ C[1]=x1^2+y1^2-1
↦ C[2]=x2+y2-1
↦ C[3]=-x1+y1+x2-y2
envelop(F,C);
↦ [1]:
↦    [1]:
↦       _[1]=(x^4+2*x^2*y^2+10*x^2*y-x^2+y^4-6*y^3+12*y^2-8*y)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=x2+y2-1
↦ [2]:
↦    [1]:
↦       _[1]=(x+y-1)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          0
↦       [2]:
↦          Special
↦       [3]:
↦          _[1]=4*y2^4-4*y2^3-y2^2
↦          _[2]=x2+y2-1
// EXAMPLE 2
// Parametric envelope
// Let c be the circle centered at the origin O(0,0) and having radius 1.
// M(x1,y1) be a mover point gliding on c.
// Let A(a0,b0) be a parametric fixed point:
// Consider the set of lines parallel to the line AO passing thoug M.
// Determine the envelope of these lines
// We let the fixed point A coordinates as free parameters of the envelope.
// We have to declare the existence of two parameters when
// defining the ring in which we call envelop,
// and set a0,b0 as the first variables of the parametric ring
// The ring is thus
if(defined(R1)){kill R1;}
ring R1=(0,a0,b0,x,y),(x1,y1),lp;
short=0;
```

```
// The lines are  F1
poly F1=b0*(x-x1)-a0*(y-y1);
// and the mover is on the circle c
ideal C1=x1^2+y1^2-1;
// The call is thus
def E1=envelop(F1,C1,"numpar",2);
E1;
↦ [1]:
↦    [1]:
↦       _[1]=(a0^2*y^2-a0^2-2*a0*b0*x*y+b0^2*x^2-b0^2)
↦    [2]:
↦       [1]:
↦          _[1]=(x^2+y^2)
↦          _[2]=(a0*y-b0*x)
↦          _[3]=(a0*x+b0*y)
↦          _[4]=(a0^2+b0^2)
↦       [2]:
↦          _[1]=(b0)
↦          _[2]=(a0)
↦    [3]:
↦       [1]:
↦          0
↦       [2]:
↦          Special
↦       [3]:
↦          _[1]=(a0^2+b0^2)*y1^2+(-a0^2)
↦          _[2]=(a0)*x1+(b0)*y1
↦ [2]:
↦    [1]:
↦       _[1]=(b0)
↦       _[2]=(a0)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          -1
↦       [2]:
↦          Accumulation
// The interesting first component  EC1 is
def EC1=E1[1][1][1];
EC1;
↦ (a0^2*y^2-a0^2-2*a0*b0*x*y+b0^2*x^2-b0^2)
// that is equivalent to  (a0*y-b0*x)^2-a0^2-b0^2.
// As expected it consists of the two lines
//    a0*y-b0*x - sqrt(a0^2+b0^2),
//    a0*y-b0*x + sqrt(a0^2+b0^2),
// parallel to the line OM passing at the
// points of the circle in the line perpendicular to OA.
// EXAMPLE 3
// Parametric envelope
// Let c be the circle centered at the origin O(a1,b1) and having radiusr,
// where a1,b1,r are fixed parameters
// M(x1,y1) be a mover point gliding on c.
// Let A(a0,b0) be a parametric fixed point:
// Consider the set of lines parallel to the line AO passing thoug M.
// Determine the envelope of these lines
// We let the fixed point A,point M and r as free parameters of the envelope.
// We have to declare the existence of 5 parameters when
// defining the ring in which we call envelop,
// and set a0,b0,a1,b1,r as the first variables of the parametric ring
// The ring is thus
if(defined(R1)){kill R1;}
ring R1=(0,a0,b0,a1,b1,r,x,y),(x1,y1),lp;
short=0;
// The lines are  F1
poly F1=b0*(x-x1)-a0*(y-y1);
// and the mover is on the circle c
ideal C1=(x1-a1)^2+(y1-b1)^2-r^2;
// The call is thus
def E1=envelop(F1,C1,"numpar",5);
E1;
↦ [1]:
↦    [1]:
↦       _[1]=(a0^2*b1^2-2*a0^2*b1*y-a0^2*r^2+a0^2*y^2-2*a0*b0*a1*b1+2*a0*b0\
   *a1*y+2*a0*b0*b1*x-2*a0*b0*x*y+b0^2*a1^2-2*b0^2*a1*x-b0^2*r^2+b0^2*x^2)
↦    [2]:
↦       [1]:
↦          _[1]=(a1^2-2*a1*x+b1^2-2*b1*y+x^2+y^2)
↦          _[2]=(a0*b1-a0*y-b0*a1+b0*x)
↦          _[3]=(a0*a1-a0*x+b0*b1-b0*y)
↦          _[4]=(a0^2+b0^2)
↦       [2]:
↦          _[1]=(b0)
↦          _[2]=(a0)
↦    [3]:
↦       [1]:
↦          0
↦       [2]:
↦          Special
↦       [3]:
↦          _[1]=(a0^2+b0^2)*y1^2+(-2*a0^2*b1-2*b0^2*b1)*y1+(a0^2*b1^2-a0^2*\
   r^2+b0^2*b1^2)
↦          _[2]=(a0)*x1+(b0)*y1+(-a0*a1-b0*b1)
↦ [2]:
↦    [1]:
↦       _[1]=(b0)
↦       _[2]=(a0)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          -1
↦       [2]:
↦          Accumulation
↦ [3]:
↦   [1]:
↦     _[1]=(r)
↦     _[2]=(a1^2-2*a1*x+b1^2-2*b1*y+x^2+y^2)
↦     _[3]=(a0*b1-a0*y-b0*a1+b0*x)
↦     _[4]=(a0*a1-a0*x+b0*b1-b0*y)
↦     _[5]=(a0^2+b0^2)
↦   [2]:
↦     [1]:
↦       _[1]=1
↦   [3]:
↦     [1]:
↦       -1
↦     [2]:
↦       Accumulation
// The interesting first component  EC1 is
def EC1=E1[1][1][1];
EC1;
↦ (a0^2*b1^2-2*a0^2*b1*y-a0^2*r^2+a0^2*y^2-2*a0*b0*a1*b1+2*a0*b0*a1*y+2*a0*\
  b0*b1*x-2*a0*b0*x*y+b0^2*a1^2-2*b0^2*a1*x-b0^2*r^2+b0^2*x^2)
// which corresponds to the product of two lines
// parallel to the line AM and intercepting the circle
// on the intersection of the line perpendicuar
// to line AM passing through A
```

D.2.4.13 locusto

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: locusto(list L);
The argument must be the output of locus or locusdg or envelop. It transforms the output into a string in standard form readable in other languages, not only Singular (Geogebra).

Return: The locus in string standard form

Note: It can only be called after computing either
- locus(F) -> locusto(locus(F))
- locusdg(locus(F)) -> locusto(locusdg(locus(F)))
- envelop(F,C) -> locusto(envelop(F,C))

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,x,y),(x1,y1),dp;
short=0;
ideal S=x1^2+y1^2-4,(y-2)*x1-x*y1+2*x,(x-x1)^2+(y-y1)^2-1;
def L=locus(S);
locusto(L);
↦ [[[(x^4+2*x^2*y^2-9*x^2+y^4-9*y^2+4*y+12)],[[1]]],[[1],[Normal],[x1^2+y1^\
  2-4]]],[[(x^2+y^2-4*y+3)],[[1]]],[[0],[Special],[y1^2-3*y1+2,x1*y1-x1,x1^\
  2+3*y1-6]]]]
locusto(locusdg(L));
↦ [[[(x^4+2*x^2*y^2-9*x^2+y^4-9*y^2+4*y+12)],[[1]]],[[1],[Relevant],[x1^2+y\
  1^2-4]]]]
```

D.2.4.14 stdlocus

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: stdlocus(ideal F)
The input ideal must be the set equations defining the locus. Calling sequence: locus(F);
The input ring must be a parametrical ideal in Q[x][u], (x=tracer variables, u=remaining variables).
(Inverts the concept of parameters and variables of the ring). Special routine for determining the locus of points of a geometrical construction. Given a parametric ideal F representing the system determining the locus of points (x) which verify certain properties, the call to stdlocus(F) determines the different irreducible components of the locus. This is a simple routine, using only standard Groebner basis computation, elimination and prime decomposition instead of using grobcov. It does not determine the taxonomy, nor the holes of the components

Return: The output is a list of the tops of the components [C_1, .. , C_n] of the locus. Each component is given its top ideal p_i.

Note: The input must be the locus system.

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,x,y),(x1,y1),dp;
short=0;
// Concoid
ideal S96=x1 ^2+y1 ^2-4,(x-2)*x1 -x*y1 +2*x,(x-x1 )^2+(y-y1 )^2-1;
stdlocus(S96);
↦ [1]:
↦    _[1]=(x^4+2*x^3+x^2*y^2-3*x^2-2*x*y^2-8*x*y-6*x+2*y^2+8*y+6)
↦ [2]:
↦    _[1]=(x^2+y^2-4*y+3)
```

D.2.4.15 AssocTanToEnv

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: AssocTanToEnv(poly F,ideal C,ideal E);
poly F must be the family of hyper-surfaces whose
envelope is analyzed. It must be defined in the ring
R=Q[x_1.,,x_n][u_1,..,u_m],
ideal C must be the ideal of restrictions
in the variables u1,..um for defining the family.
C must contain less polynomials than m.
ideal E must be a component of
envelop(F,C), previously computed.
(x_1,..,x_n) are the variables of the hypersurfaces
of F, that are considered as parameters of the
parametric ring. (u_1,..,u_m) are the parameteres
of the hyper-surfaces, that are considered as variables of the parametric ring. Having computed an envelop
component E of a family of hyper-surfaces F,

with constraints C, it returns the parameter values
of the associated tangent hyper-surface of the
family passing at one point of the envelop component E. Calling sequence: (s<m)
ring R=(0,x_1,..,x_n),(u_1,..,u_m),lp;
poly F=F(x_1,..,x_n,u_1,..,u_m);
ideal C=g_1(u_1,..u_m),..,g_s(u_1,..u_m);
poly E(x_1,..,x_n);
AssocTanToEnv(F,C,E,[,options]);

Return: list [lpp,basis,segment]. The basis determines
the associated tangent hyper-surface at a point of
the envelop component E. The segment is given in Prep. See book
A. Montes. "The Groebner Cover":

Options: "moreinfo",n n=0 is the default option, and
only the segment of the top of the component is shown. n=1 makes the result to shown
all the segments.

Note: grobcov is called internally.

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,x,y),(r,s,y1,x1),lp;
poly F=(x-x1)^2+(y-y1)^2-r;
ideal g=(x1-2*(s+r))^2+(y1-s)^2-s;
def E=envelop(F,g);
E;
↦ [1]:
↦    [1]:
↦       _[1]=(512*x^3-1024*x^2*y^2-2560*x^2*y-640*x^2+4096*x*y^3+4864*x*y^2\
   -704*x*y+984*x-4096*y^4+1536*y^3-16*y^2+144*y+289)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=1024*y1^4-1024*y1^3*x1+896*y1^3+256*y1^2*x1^2-1344*y1^2*x1+\
   132*y1^2+128*y1*x1^2-560*y1*x1-28*y1+16*x1^2-72*x1+1
def A=AssocTanToEnv(F,g,E[1][1][1]);
A;
↦ [1]:
↦    _[1]=x1
↦    _[2]=y1
↦    _[3]=s
↦    _[4]=r
↦ [2]:
↦    _[1]=4*x1+(-4*x-1)
↦    _[2]=(1024*y^3-768*y^2+192*y-556)*y1+(192*x^2+128*x*y^2-832*x*y-136*x-\
   1024*y^4+1024*y^3+272*y^2+48*y-29)
```

```
↦     _[3]=(5120*y^3-3840*y^2+960*y-2780)*s+(1024*x^2*y+704*x^2-2048*x*y^3-1\
   920*x*y^2-4288*x*y+984*x-1024*y^4+4352*y^3-816*y^2-880*y+357)
↦     _[4]=(2048*y^3-1536*y^2+384*y-1112)*r+(-512*x^2*y-256*x^2+1792*x*y^2+1\
   536*x*y-4*x-2176*y^3+928*y^2+368*y+85)
↦ [3]:
↦     [1]:
↦        [1]:
↦           _[1]=(512*x^3-1024*x^2*y^2-2560*x^2*y-640*x^2+4096*x*y^3+4864*x*\
   y^2-704*x*y+984*x-4096*y^4+1536*y^3-16*y^2+144*y+289)
↦        [2]:
↦           [1]:
↦              _[1]=(256*y^3-192*y^2+48*y-139)
↦              _[2]=(24*x-64*y^2-16*y-13)
def M1=coef(A[2][1],x1);
def M2=coef(A[2][2],y1);
def M3=coef(A[2][3],s);
def M4=coef(A[2][4],r);
"x1=";-M1[2,2]/M1[2,1];
↦ x1=
↦ (4*x+1)/4
"y1=";-M2[2,2]/M2[2,1];
↦ y1=
↦ (-192*x^2-128*x*y^2+832*x*y+136*x+1024*y^4-1024*y^3-272*y^2-48*y+29)/(102\
   4*y^3-768*y^2+192*y-556)
"s=";-M3[2,2]/M3[2,1];
↦ s=
↦ (-1024*x^2*y-704*x^2+2048*x*y^3+1920*x*y^2+4288*x*y-984*x+1024*y^4-4352*y\
   ^3+816*y^2+880*y-357)/(5120*y^3-3840*y^2+960*y-2780)
"r=";-M4[2,2]/M4[2,1];
↦ r=
↦ (512*x^2*y+256*x^2-1792*x*y^2-1536*x*y+4*x+2176*y^3-928*y^2-368*y-85)/(20\
   48*y^3-1536*y^2+384*y-1112)
```

D.2.4.16 FamElemsAtEnvCompPoints

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: FamElemsAtEnvCompPoints(poly F,ideal C,poly E);
poly F must be the family of hyper-surfaces whose
envelope is analyzed. It must be defined in the ring
R=Q[x_1.,,x_n][u_1,..,u_m],
ideal C must be the ideal of restrictions on the
variables u1,..um.
Must contain less polynomials than m.
ideal E must be a component of
envelop(F,C), previously computed.
After computing the envelop of a family of
hyper-surfaces F, with constraints C,
Consider a component with top E. The call to
FamElemsAtEnvCompPoints(F,C,E)
returns the parameter values of the
set of all hyper-surfaces of the family passing at
one point of the envelop component E.

Calling sequence:
ring R=(0,x_1,..,x_n),(u_1,..,u_m),lp;
poly F=F(x_1,..,x_n,u_1,..,u_m);
ideal C=g_1(u_1,..u_m),..,g_s(u_1,..u_m);
poly E(x_1,..,x_n);
FamElemsAtEnvCompPoints(F,C,E[,options]);

Return: list [lpp,basis,segment]. The basis determines
the parameter values of the of hyper-surfaces that
pass at a fixed point of the envelop component E.
The lpp determines the dimension of the set.
The segment is the component and is given in Prep.
Fixing the values of (x_1,..,x_n) inside E, the basis allows to detemine the values of the parameters
(u_1,..u_m), of the hyper-surfaces passing at a point of E. See the book
A. Montes. "The Groebner Cover" (Discussing
Parametric Polynomial Systems).

Options: "moreinfo",n n=0 is the default option, and
only the segment of the top of the component is shown. n=1 makes the result to shown all the segments.

Note: grobcov is called internally.
The basering R, must be of the form Q[a][x]
(a=parameters, x=variables).

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,x,y),(t),dp;
short=0;
poly F=(x-5*t)^2+y^2-9*t^2;
ideal C;
def Env=envelop(F,C);
Env;
↦ [1]:
↦    [1]:
↦       _[1]=(3*x+4*y)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=0
↦ [2]:
↦    [1]:
↦       _[1]=(3*x-4*y)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=0
// E is a component of the envelope:
def E=Env[1][1][1];
E;
↦ (3*x+4*y)
def A=AssocTanToEnv(F,C,E);
A;
↦ [1]:
↦    _[1]=t
↦ [2]:
↦    _[1]=12*t+(5*y)
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=(3*x+4*y)
↦       [2]:
↦          [1]:
↦             _[1]=1
// The basis of the parameter values of the associated
//    tangent component is
A[2][1];
↦ 12*t+(5*y)
// Thus t=-(5/12)*y, and  the associated tangent family
//    element at (x,y) is
subst(F,t,-(5/12)*y);
↦ (18*x^2+75*x*y+68*y^2)/18
def FE=FamElemsAtEnvCompPoints(F,C,E);
FE;
↦ [1]:
↦    _[1]=t^2
↦ [2]:
↦    _[1]=144*t^2+(120*y)*t+(25*y^2)
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=(3*x+4*y)
↦       [2]:
↦          [1]:
↦             _[1]=1
factorize(FE[2][1]);
↦ [1]:
↦    _[1]=1
↦    _[2]=12*t+(5*y)
↦ [2]:
↦    1,2
// Thus the unique family element passing through the envelope point (x,y)
// corresponds to the value of t of the Associated Tangent
// EXAMPLE:
if(defined(R)){kill R;}
ring R=(0,x,y),(r,s,y1,x1),lp;
poly F=(x-x1)^2+(y-y1)^2-r;
ideal g=(x1-2*(s+r))^2+(y1-s)^2-s;
def E=envelop(F,g);
E;
↦ [1]:
↦    [1]:
↦       _[1]=(512*x^3-1024*x^2*y^2-2560*x^2*y-640*x^2+4096*x*y^3+4864*x*y^2\
   -704*x*y+984*x-4096*y^4+1536*y^3-16*y^2+144*y+289)
↦    [2]:
↦       [1]:
↦          _[1]=1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          Normal
↦       [3]:
↦          _[1]=1024*y1^4-1024*y1^3*x1+896*y1^3+256*y1^2*x1^2-1344*y1^2*x1+\
   132*y1^2+128*y1*x1^2-560*y1*x1-28*y1+16*x1^2-72*x1+1
def A=AssocTanToEnv(F,g,E[1][1][1]);
A;
↦ [1]:
↦    _[1]=x1
↦    _[2]=y1
↦    _[3]=s
↦    _[4]=r
↦ [2]:
↦    _[1]=4*x1+(-4*x-1)
↦    _[2]=(1024*y^3-768*y^2+192*y-556)*y1+(192*x^2+128*x*y^2-832*x*y-136*x-\
   1024*y^4+1024*y^3+272*y^2+48*y-29)
↦    _[3]=(5120*y^3-3840*y^2+960*y-2780)*s+(1024*x^2*y+704*x^2-2048*x*y^3-1\
   920*x*y^2-4288*x*y+984*x-1024*y^4+4352*y^3-816*y^2-880*y+357)
↦    _[4]=(2048*y^3-1536*y^2+384*y-1112)*r+(-512*x^2*y-256*x^2+1792*x*y^2+1\
   536*x*y-4*x-2176*y^3+928*y^2+368*y+85)
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=(512*x^3-1024*x^2*y^2-2560*x^2*y-640*x^2+4096*x*y^3+4864*x*\
   y^2-704*x*y+984*x-4096*y^4+1536*y^3-16*y^2+144*y+289)
↦       [2]:
↦          [1]:
↦             _[1]=(256*y^3-192*y^2+48*y-139)
↦             _[2]=(24*x-64*y^2-16*y-13)
def M1=coef(A[2][1],x1);
def M2=coef(A[2][2],y1);
def M3=coef(A[2][3],s);
def M4=coef(A[2][4],r);
// The parameter values corresponding to the family
//    element tangent at point (x,y) of the envelope are:
"x1=";-M1[2,2]/M1[2,1];
```

```
↦ x1=
↦ (4*x+1)/4
"y1=";-M2[2,2]/M2[2,1];
↦ y1=
↦ (-192*x^2-128*x*y^2+832*x*y+136*x+1024*y^4-1024*y^3-272*y^2-48*y+29)/(102\
   4*y^3-768*y^2+192*y-556)
"s=";-M3[2,2]/M3[2,1];
↦ s=
↦ (-1024*x^2*y-704*x^2+2048*x*y^3+1920*x*y^2+4288*x*y-984*x+1024*y^4-4352*y\
   ^3+816*y^2+880*y-357)/(5120*y^3-3840*y^2+960*y-2780)
"r=";-M4[2,2]/M4[2,1];
↦ r=
↦ (512*x^2*y+256*x^2-1792*x*y^2-1536*x*y+4*x+2176*y^3-928*y^2-368*y-85)/(20\
   48*y^3-1536*y^2+384*y-1112)
// Now detect if there are other family elements passing at this point:
def FE=FamElemsAtEnvCompPoints(F,g,E[1][1][1]);
FE;
↦ [1]:
↦    _[1]=s^2
↦    _[2]=r
↦ [2]:
↦    _[1]=2560*s^2+4096*s*y1^2+(-8192*y-1024)*s*y1+4096*s*x1^2+(-8192*x-204\
   8)*s*x1+(4096*x^2+4096*y^2-512)*s+2048*y1^4+(-8192*y)*y1^3+4096*y1^2*x1^2\
   +(-8192*x-2048)*y1^2*x1+(4096*x^2+12288*y^2+512)*y1^2+(-8192*y)*y1*x1^2+(\
   16384*x*y+4096*y)*y1*x1+(-8192*x^2*y-8192*y^3)*y1+2048*x1^4+(-8192*x-2048\
   )*x1^3+(12288*x^2+4096*x+4096*y^2+512)*x1^2+(-16384*x^2*y^2-40960*x^2*y-1\
   2288*x^2+65536*x*y^3+69632*x*y^2-11264*x*y+15744*x-65536*y^4+24576*y^3-23\
   04*y^2+2304*y+4624)*x1+(8192*x^2*y^4+24576*x^2*y^3+46080*x^2*y^2+28416*x^\
   2*y-736*x^2-32768*x*y^5-104448*x*y^4-118272*x*y^3-18048*x*y^2-16736*x*y-6\
   076*x+32768*y^6+69632*y^5-8064*y^4-8512*y^3-5112*y^2-6500*y-1445)
↦    _[2]=r-y1^2+(2*y)*y1-x1^2+(2*x)*x1+(-x^2-y^2)
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=(512*x^3-1024*x^2*y^2-2560*x^2*y-640*x^2+4096*x*y^3+4864*x*\
   y^2-704*x*y+984*x-4096*y^4+1536*y^3-16*y^2+144*y+289)
↦       [2]:
↦          [1]:
↦             _[1]=1
// FE[1] is the set of lpp. It has dimension 4-2=2.
//     Thus there are points of the envelope at which
//     they pass infinitely many circles of the family.
//     To separe the points of the envelope further analysis must be done.
```

D.2.4.17 discrim

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: discrim(f,x);
poly f: the polynomial in Q[a][x] or Q[x] of degree 2 in x poly x: can be a variable or a parameter of the ring.

Return: the factorized discriminant of f wrt x for discussing its sign

Example:

```
LIB "grobcov.lib";
if(defined(R)){kill R;}
ring R=(0,a,b,c),(x,y),dp;
short=0;
poly f=a*x^2*y+b*x*y+c*y;
discrim(f,x);
↦ [1]:
↦    _[1]=-1
↦    _[2]=(4*a*c-b^2)
↦    _[3]=y
↦ [2]:
↦    1,1,2
```

D.2.4.18 WLemma

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: WLemma(F,A[,options]);
The first argument ideal F in Q[x_1,..,x_n][u_1,..,u_m]; The second argument ideal A in Q[x_1,..,x_n].
Calling sequence:
ring R=(0,x_1,..,x_n),(u_1,..,u_m),lp;
ideal F=f_1(x_1,..,x_n,u_1,..,u_m),..,
f_s(x_1,..,x_n,u_1,..,u_m);
ideal A=g_1(u_1,..u_m),..,g_s(u_1,..u_m);
list # : Options
Calling sequence:
WLemma(F,A[,options]);

Given the ideal F and ideal A
it returns the list (lpp,B,S) were B is the
reduced Groebner basis of the specialized F over
the segment S, subset of V(A) with top A,
determined by Wibmer's Lemma.
S is determined in P-representation
(or optionally in C-representation). The basis is
given by I-regular functions.

Options: either ("rep", 0) or ("rep",1) the representation of the resulting segment, by default is 0 =P-representation, (default) but can be set to
1=C-representation.

Return: list of [lpp,B,S] =
[leading power product, basis,segment],
being B the reduced Groebner Basis given by
I-regular functions in full representation, of
the specialized ideal F on the segment S,
subset of V(A) with top A.
given in P- or C-representation.
It is the result of Wibmer's Lemma. See
A. Montes , M. Wibmer, "Groebner Bases for
Polynomial Systems with parameters".
JSC 45 (2010) 1391-1425.)
or the book

A. Montes. "The Groebner Cover" (Discussing Parametric Polynomial Systems).

Note: The basering R, must be of the form Q[a][x] (a=parameters, x=variables).

Example:

```
LIB "grobcov.lib";
if(defined(RE)){kill RE;}
ring RE=(0,a,b,c,d,e,f),(x,y),lp;
ideal F=a*x^2+b*x*y+c*y^2,d*x^2+e*x*y+f*y^2;
ideal A=a*e-b*d;
WLemma(F,A);
↦ [1]:
↦    _[1]=y2
↦    _[2]=x2
↦ [2]:
↦    [1]:
↦       _[1]=y2
↦    [2]:
↦       _[1]=(d)*x2+(e)*xy+(f)*y2
↦       _[2]=(a)*x2+(b)*xy+(c)*y2
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=(ae-bd)
↦       [2]:
↦          [1]:
↦             _[1]=(bf-ce)
↦             _[2]=(af-cd)
↦             _[3]=(ae-bd)
↦          [2]:
↦             _[1]=(d)
↦             _[2]=(a)
WLemma(F,A,"rep",1);
↦ [1]:
↦    _[1]=y2
↦    _[2]=x2
↦ [2]:
↦    [1]:
↦       _[1]=y2
↦    [2]:
↦       _[1]=(d)*x2+(e)*xy+(f)*y2
↦       _[2]=(a)*x2+(b)*xy+(c)*y2
↦ [3]:
↦    [1]:
↦       _[1]=(ae-bd)
↦    [2]:
↦       _[1]=(bdf-cde)
↦       _[2]=(af-cd)
↦       _[3]=(ae-bd)
```

D.2.4.19 WLCGS

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

D.2.4.20 intersectpar

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: interectpar(list of ideals S)
list S=ideal I1,...,ideal Ik;

Return: The intersection of the ideals I1 ... Ik in Q[x,a]

Note: The routine is called in Q[a][x],
The ideals I1,..,Ik can be ideals depending only on [a] or on [x,a]

Example:

```
LIB "grobcov.lib";
"EAXMPLE:";
↦ EAXMPLE:
if(defined(R)){kill R;}
ring R=(0,x,y,z),(x1,y1,z1),lp;
ideal I1=x+y*z*x1;
ideal I2=x-y*z*y1;
ideal I3=x+y+z*z1;
list S=I1,I2,I3;
S;
↦ [1]:
↦    _[1]=(y*z)*x1+(x)
↦ [2]:
↦    _[1]=(-y*z)*y1+(x)
↦ [3]:
↦    _[1]=(z)*z1+(x+y)
intersectpar(S);
↦ _[1]=(y^2*z^3)*x1*y1*z1+(x*y^2*z^2+y^3*z^2)*x1*y1+(-x*y*z^2)*x1*z1+(-x^2*\
   y*z-x*y^2*z)*x1+(x*y*z^2)*y1*z1+(x^2*y*z+x*y^2*z)*y1+(-x^2*z)*z1+(-x^3-x^\
   2*y)
```

D.2.4.21 ADGT

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: ADGT(ideal H, ideal T, ideal H1,ideal T1[,options]);
H: ideal in Q[a][x] hypothesis
T: ideal in Q[a][x] thesis
H1: ideal in Q[a][x] negative hypothesis, only dependent on [a] T1: ideal in Q[a][x] negative thesis
of the Proposition (H and not H1) => (T and not T1)

Return: The list [[1,S1],[2,S2],..],
S1, S2, .. represent the set of parameter values
that must be verified as supplementary
conditions for the Proposition to become a Theorem.
They are given by default in Prep.
If the proposition is generally true,

(the proposition is already a theorem), then
the generic segment of the internal grobcov
called is also returned to provide information
about the values of the variables determined
for every value of the parameters.
If the proposition is false for every values of the
parameters, then the empty list is returned.

Options: An option is a pair of arguments: string,
integer. To modify the default options, pairs
of arguments -option name, value- of valid options
must be added to the call.
Option "rep",0-1: The default is ("rep",0)
and then the segments are given in canonical
P-representation.
Option ("rep",1) represents the segments
in canonical C-representation,
Option "gseg",0-1: The default is "gseg",1
and then when the proposition is generally true,
ADGT outputs a second element which is the
"generic segment" to provide supplementary information. With option "gseg",0 this is avoided.
Option "neg", 0,1: The default is "neg",0
With option "neg",0 Rabinovitch trick is used for negative hypothesis and thesis With option "neg",1 Difference of constructible sets is used instead.

Note: The basering R, must be of the form Q[a][x],
(a=parameters, x=variables), and
should be defined previously. The ideals
must be defined on R.

Example:

```
LIB "grobcov.lib";
// Determine the supplementary conditions
// for the non-degenerate  triangle A(-1,0), B(1,0), C(x,y)
// to have an ortic non-degenerate isosceles triangle
if(defined(R)){kill R;}
ring R=(0,x,y),(x2,y2,x1,y1),lp;
// Hypothesis H: the triangle  A1(x1,y1), B1(x2,y2), C1(x,0), is the
// orthic triangle of ABC
ideal H=-y*x1+(x-1)*y1+y,
(x-1)*(x1+1)+y*y1,
-y*x2+(x+1)*y2-y,
(x+1)*(x2-1)+y*y2;
// Thesis T: the orthic triangle is isosceles
ideal T=(x1-x)^2+y1^2-(x2-x)^2-y2^2;
// Negative Hypothesis H1: ABC is non-degenerate
ideal H1=y;
// Negative Thesis T1: the orthic triangle is non-degenerate
ideal T1=x*(y1-y2)-y*(x1-x2)+x1*y2-x2*y1;
// Complementary conditions for the
// Proposition (H and not H1) => (T and not T1)
// to be true
```

```
ADGT(H,T,H1,T1);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          [1]:
↦             _[1]=(x^2-y^2-1)
↦          [2]:
↦             [1]:
↦                _[1]=(y)
↦                _[2]=(x-1)
↦             [2]:
↦                _[1]=(y)
↦                _[2]=(x+1)
↦             [3]:
↦                _[1]=(y^2+1)
↦                _[2]=(x)
↦       [2]:
↦          [1]:
↦             _[1]=(x)
↦          [2]:
↦             [1]:
↦                _[1]=(y)
↦                _[2]=(x)
↦             [2]:
↦                _[1]=(y-1)
↦                _[2]=(x)
↦             [3]:
↦                _[1]=(y+1)
↦                _[2]=(x)
↦             [4]:
↦                _[1]=(y^2+1)
↦                _[2]=(x)
// Now using diference of constructible sets for negative hypothesis and thesis
ADGT(H,T,H1,T1,"neg",1);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          [1]:
↦             _[1]=(x^2-y^2-1)
↦          [2]:
↦             [1]:
↦                _[1]=(y)
↦                _[2]=(x-1)
↦             [2]:
↦                _[1]=(y)
↦                _[2]=(x+1)
↦             [3]:
↦                _[1]=(y^2+1)
↦                _[2]=(x)
↦       [2]:
↦          [1]:
↦             _[1]=(x)
↦          [2]:
↦             [1]:
↦                _[1]=(y)
↦                _[2]=(x)
↦             [2]:
↦                _[1]=(y-1)
↦                _[2]=(x)
↦             [3]:
↦                _[1]=(y+1)
↦                _[2]=(x)
↦             [4]:
↦                _[1]=(y^2+1)
↦                _[2]=(x)
// The results are identical using both methods for the negative propositions
// - Rabinovitch or
// - DifConsLCSets
// EXAMPLE 2
// Automatic Theorem Proving.
// The nine points circle theorem.
// Vertices of the triangle: A(-2,0), B(2,0), C(2a,2b)
// Heigth foot: A1(x1,y1),
// Heigth foot: B1(x2,y2),
// Heigth foot: C1(2a,0)
// Middle point BC:  A2(a+1,b)
// Middle point CA:  B2 (a-1,b)
// Middle point AB:  C2(0,0)
// Ortocenter:   O(2x0,2y0)
// Middle point of A and O:  A3(x0-1,y0)
// Middle point of B and O:  B3(x0+1,y0)
// Middle point of C and O:  C3(x0+a,y0+b)
// Nine points circle: c:=(X-x3)^2+(Y-y3)^2-r2
if (defined(R1)){kill R1;}
ring R1=(0,a,b),(x1,y1,x2,y2,x0,y0,x3,y3,r2),dp;
short=0;
ideal H=-x1*b+(a-1)*y1+2*b,
(a-1)*x1+b*y1+2*a-2,
-x2*b+(a+1)*y2-2*b,
(a+1)*x2+b*y2-2*a-2,
-x0*y1+(x1+2)*y0-y1,
-x0*y2+(x2-2)*y0+y2;
ideal T=(x1-2*x3)^2+(y1-2*y3)^2-r2,
(a+1-2*x3)^2+(b-2*y3)^2-r2,
(x0-1-2*x3)^2+(y0-2*y3)^2-r2,
(x2-2*x3)^2+(y2-2*y3)^2-r2,
(a-1-2*x3)^2+(b-2*y3)^2-r2,
(x0+1-2*x3)^2+(y0-2*y3)^2-r2,
(2*a-2*x3)^2+4*y3^2-r2,
4*x3^2+4*y3^2-r2,
(x0+a-2*x3)^2+(y0+b-2*y3)^2-r2;
ADGT(H,T,b,1);
```

```
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             [1]:
↦                _[1]=(a^2+2*a+b^2+1)
↦             [2]:
↦                _[1]=(a^2-2*a+b^2+1)
↦             [3]:
↦                _[1]=(b)
↦ [2]:
↦    [1]:
↦       Generic segment
↦    [2]:
↦       [1]:
↦          _[1]=r2
↦          _[2]=y3
↦          _[3]=x3
↦          _[4]=y0
↦          _[5]=x0
↦          _[6]=y2
↦          _[7]=x2
↦          _[8]=y1
↦          _[9]=x1
↦       [2]:
↦          _[1]=(4*b^2)*r2+(-a^4-2*a^2*b^2+2*a^2-b^4-2*b^2-1)
↦          _[2]=(4*b)*y3+(a^2-b^2-1)
↦          _[3]=2*x3+(-a)
↦          _[4]=(b)*y0+(a^2-1)
↦          _[5]=x0+(-a)
↦          _[6]=(a^2+2*a+b^2+1)*y2+(-4*a*b-4*b)
↦          _[7]=(a^2+2*a+b^2+1)*x2+(-2*a^2-4*a+2*b^2-2)
↦          _[8]=(a^2-2*a+b^2+1)*y1+(4*a*b-4*b)
↦          _[9]=(a^2-2*a+b^2+1)*x1+(2*a^2-4*a-2*b^2+2)
↦       [3]:
↦          [1]:
↦             [1]:
↦                _[1]=0
↦             [2]:
↦                [1]:
↦                   _[1]=(a^2+2*a+b^2+1)
↦                [2]:
↦                   _[1]=(a^2-2*a+b^2+1)
↦                [3]:
↦                   _[1]=(b)
// Thus the nine point circle theorem is true for all real points excepting b=0.
```

D.2.4.22 ConsLevels

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: ConsLevels(list L);
L=[[P1,Q1],...,[Ps,Qs]] is a list of lists of of pairs of ideals represening the constructible set
S=V(P1) \ V(Q1) u ... u V(Ps) \ V(Qs).
To be called in a ring Q[a][x] or a ring Q[a]. But the ideals can contain only the parameters in Q[a].

Return: The list of ideals [a1,a2,...,at] representing the
closures of the canonical levels of S and its
complement C wrt to the closure of S. The
canonical levels of S are represented by theirs
Crep. So we have:
Levels of S: [a1,a2],[a3,a4],...
Levels of C: [a2,a3],[a4,a5],...
S=V(a1) \ V(a2) u V(a3) \ V(a4) u ...
C=V(a2 \ V(a3) u V(a4) \ V(a5) u ...
The expression of S can be obtained from the
output of ConsLevels by
the call to Levels.

Note: The algorithm was described in
J.M. Brunat, A. Montes. "Computing the canonical
representation of constructible sets."
Math. Comput. Sci. (2016) 19: 165-178.

Example:

```
LIB "grobcov.lib";
// EXAMPLE:
if(defined(R)){kill R;}
ring R=0,(x,y,z),lp;
short=0;
ideal P1=(x^2+y^2+z^2-1);
ideal Q1=z,x^2+y^2-1;
ideal P2=y,x^2+z^2-1;
ideal Q2=z*(z+1),y,x*(x+1);
ideal P3=x;
ideal Q3=5*z-4,5*y-3,x;
list Cr1=Crep(P1,Q1);
list Cr2=Crep(P2,Q2);
list Cr3=Crep(P3,Q3);
list L=list(Cr1,Cr2,Cr3);
L;
↦ [1]:
↦    [1]:
↦       _[1]=x^2+y^2+z^2-1
↦    [2]:
↦       _[1]=z
↦       _[2]=x^2+y^2-1
↦ [2]:
↦    [1]:
```

```
↦       _[1]=y
↦       _[2]=x^2+z^2-1
↦    [2]:
↦       _[1]=z^2+z
↦       _[2]=y
↦       _[3]=x+z+1
↦ [3]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=5*z-4
↦       _[2]=5*y-3
↦       _[3]=x
def LL=ConsLevels(L);
LL;
↦ [1]:
↦    _[1]=x^3+x*y^2+x*z^2-x
↦ [2]:
↦    _[1]=z
↦    _[2]=x^2+y^2-1
↦ [3]:
↦    _[1]=z
↦    _[2]=x+y^2-1
↦    _[3]=x*y
↦    _[4]=x^2-x
↦ [4]:
↦    _[1]=1
def LLL=Levels(LL);
LLL;
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          _[1]=x^3+x*y^2+x*z^2-x
↦       [2]:
↦          _[1]=z
↦          _[2]=x^2+y^2-1
↦ [2]:
↦    [1]:
↦       3
↦    [2]:
↦       [1]:
↦          _[1]=z
↦          _[2]=x+y^2-1
↦          _[3]=x*y
↦          _[4]=x^2-x
↦       [2]:
↦          _[1]=1
```

D.2.4.23 Levels

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: Levels(list L);
The input list L must be the output of the call to the routine ConsLevels of a constructible set:
L=[a1,a2,..,ak], where the a's are the closures
of the levels, determined by ConsLevels.
Levels selects the levels of the
constructible set. To be called in a ring Q[a][x]
or a ring Q[a]. But the ideals can contain
only the parameters in Q[a].

Return: The levels of the constructible set:
Lc=[[1,[a1,a2]],[3,[a3,a4]],..,
[2l-1,[a_{2l-1},a_{2l}]]]
the list of levels of S

Example:

```
LIB "grobcov.lib";
// EXAMPLE:
if(defined(R)){kill R;}
ring R=0,(x,y,z),lp;
short=0;
ideal P1=(x^2+y^2+z^2-1);
ideal Q1=z,x^2+y^2-1;
ideal P2=y,x^2+z^2-1;
ideal Q2=z*(z+1),y,x*(x+1);
ideal P3=x;
ideal Q3=5*z-4,5*y-3,x;
list Cr1=Crep(P1,Q1);
list Cr2=Crep(P2,Q2);
list Cr3=Crep(P3,Q3);
list L=list(Cr1,Cr2,Cr3);
L;
↦ [1]:
↦    [1]:
↦       _[1]=x^2+y^2+z^2-1
↦    [2]:
↦       _[1]=z
↦       _[2]=x^2+y^2-1
↦ [2]:
↦    [1]:
↦       _[1]=y
↦       _[2]=x^2+z^2-1
↦    [2]:
↦       _[1]=z^2+z
↦       _[2]=y
↦       _[3]=x+z+1
↦ [3]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=5*z-4
↦       _[2]=5*y-3
↦       _[3]=x
```

```
def LL=ConsLevels(L);
LL;
↦ [1]:
↦    _[1]=x^3+x*y^2+x*z^2-x
↦ [2]:
↦    _[1]=z
↦    _[2]=x^2+y^2-1
↦ [3]:
↦    _[1]=z
↦    _[2]=x+y^2-1
↦    _[3]=x*y
↦    _[4]=x^2-x
↦ [4]:
↦    _[1]=1
def LLL=Levels(LL);
LLL;
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          _[1]=x^3+x*y^2+x*z^2-x
↦       [2]:
↦          _[1]=z
↦          _[2]=x^2+y^2-1
↦ [2]:
↦    [1]:
↦       3
↦    [2]:
↦       [1]:
↦          _[1]=z
↦          _[2]=x+y^2-1
↦          _[3]=x*y
↦          _[4]=x^2-x
↦       [2]:
↦          _[1]=1
```

D.2.4.24 Grob1Levels

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Usage: Grob1Levels(list G);
G is the output of grobcov(F,"rep",1)
for obtaining the segments in C-rep.
Then Grob!Levels, selects the set of segments S of G having solutions (i.e. with basis different from 1), and determines the canonical levels of this constructible set.
To be called in a ring Q[a][x].

Return: The list of ideals
[a1,a2,...,at]
representing the closures of the canonical levels of S and its complement C wrt to the closure of S.

The levels of S and C are
Levels of S: [a1,a2],[a3,a4],...

Levels of C: [a2,a3],[a4,a5],...
S=V(a1) \ V(a2) u V(a3) \ V(a4) u ...
C=V(a2 \ V(a3) u V(a4) \ V(a5) u ...
The expression of S can be obtained from the
output of Grob1Levels by
the call to Levels.

Note: The algorithm was described in
J.M. Brunat, A. Montes. "Computing the canonical
representation of constructible sets."
Math. Comput. Sci. (2016) 19: 165-178.

Example:

```
LIB "grobcov.lib";
"EAXMPLE:";
↦ EAXMPLE:
if (defined(R)) {kill R;}
ring R=(0,x,y),(x1,y1,x2,y2),lp;
ideal F=-y*x1+(x-1)*y1+y,
(x-1)*(x1+1)+y*y1,
-y*x2+(x+1)*y2-y,
(x+1)*(x2-1)+y*y2,
(x1-x)^2+y1^2-(x1-x)^2-y2^2;
def G=grobcov(F,"rep",1);
G;
↦ [1]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=(x^5*y-2*x^3*y-x*y^5+x*y)
↦ [2]:
↦    [1]:
↦       _[1]=y2
↦       _[2]=x2
↦       _[3]=y1
↦       _[4]=x1
↦    [2]:
↦       _[1]=(x^5+2*x^4*y^6+4*x^4*y^4+2*x^4*y^2+4*x^3*y^6+8*x^3*y^4+4*x^3*y\
   ^2-2*x^3-4*x^2*y^6-8*x^2*y^4-4*x^2*y^2-4*x*y^6-9*x*y^4-4*x*y^2+x-2*y^10-4\
   *y^8+4*y^4+2*y^2)*y2+(-6*x^4*y^5-6*x^4*y^3-2*x^3*y^7-4*x^3*y^5-2*x^3*y^3-\
   2*x^2*y^7+8*x^2*y^5+10*x^2*y^3+2*x*y^9+6*x*y^7+6*x*y^5+2*x*y^3+4*y^9+4*y^\
   7-4*y^5-4*y^3)
↦       _[2]=(x^5+2*x^4*y^7+4*x^4*y^5+2*x^4*y^3+4*x^3*y^7+8*x^3*y^5+4*x^3*y\
   ^3-2*x^3-4*x^2*y^7-8*x^2*y^5-4*x^2*y^3-4*x*y^7-8*x*y^5-x*y^4-4*x*y^3+x-2*\
   y^11-4*y^9+4*y^5+2*y^3)*x2+(-x^5-6*x^4*y^5-6*x^4*y^3+2*x^3+2*x^2*y^9+6*x^\
   2*y^5+8*x^2*y^3+x*y^4-x-2*y^11+4*y^7-2*y^3)
↦       _[3]=(x^5+2*x^4*y^6+4*x^4*y^4+2*x^4*y^2+4*x^3*y^6+8*x^3*y^4+4*x^3*y\
   ^2-2*x^3-4*x^2*y^6-8*x^2*y^4-4*x^2*y^2-4*x*y^6-9*x*y^4-4*x*y^2+x-2*y^10-4\
   *y^8+4*y^4+2*y^2)*y1+(-2*x^4*y^5-2*x^4*y^3-2*x^3*y^7-4*x^3*y^5-2*x^3*y^3+\
   2*x^2*y^7+8*x^2*y^5+6*x^2*y^3+2*x*y^9+6*x*y^7+6*x*y^5+2*x*y^3+4*y^9+4*y^7\
   -4*y^5-4*y^3)
↦       _[4]=(x^5+2*x^4*y^7+4*x^4*y^5+2*x^4*y^3+4*x^3*y^7+8*x^3*y^5+4*x^3*y\
   ^3-2*x^3-4*x^2*y^7-8*x^2*y^5-4*x^2*y^3-4*x*y^7-8*x*y^5-x*y^4-4*x*y^3+x-2*\
   y^11-4*y^9+4*y^5+2*y^3)*x1+(x^5-4*x^4*y^7-6*x^4*y^5-2*x^4*y^3-2*x^3+2*x^2\
   *y^9+8*x^2*y^7+6*x^2*y^5-x*y^4+x+2*y^11-4*y^7+2*y^3)
↦    [3]:
↦       [1]:
↦          _[1]=(x^5*y-2*x^3*y-x*y^5+x*y)
↦       [2]:
↦          _[1]=(x*y)
↦          _[2]=(x^2-y^2-1)
↦          _[3]=(y^3+y)
↦ [3]:
↦    [1]:
↦       _[1]=y2^2
↦       _[2]=y1
↦       _[3]=x1
↦    [2]:
↦       _[1]=y2^2
↦       _[2]=y1
↦       _[3]=x1+1
↦    [3]:
↦       [1]:
↦          _[1]=(y)
↦          _[2]=(x+1)
↦       [2]:
↦          _[1]=1
↦ [4]:
↦    [1]:
↦       _[1]=y2
↦       _[2]=x2
↦       _[3]=y1^2
↦    [2]:
↦       _[1]=y2
↦       _[2]=x2-1
↦       _[3]=y1^2
↦    [3]:
↦       [1]:
↦          _[1]=(y)
↦          _[2]=(x-1)
↦       [2]:
↦          _[1]=1
↦ [5]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       [1]:
↦          _[1]=(x)
↦          _[2]=(y^2+1)
↦         [2]:
↦            _[1]=1
def L=Grob1Levels(G);
L;
↦ [1]:
↦    _[1]=(x^5*y-2*x^3*y-x*y^5+x*y)
↦ [2]:
↦    _[1]=(x)
↦    _[2]=(y^2+1)
↦ [3]:
↦    _[1]=1
def LL=Levels(L);
LL;
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          _[1]=(x^5*y-2*x^3*y-x*y^5+x*y)
↦       [2]:
↦          _[1]=(x)
↦          _[2]=(y^2+1)
```

D.2.4.25 DifConsLCSets

Procedure from library `grobcov.lib` (see Section D.2.4 [grobcov_lib], page 817).

Return: A list of locally closed sets equivalent to the difference S= A "" B. Lc=[[1][p1,q1]] [[2][p2,q2]]..],
For obtaining the canonical representation into levels of the constructible A "" B one have to apply ConsLevels and then optatively Levels.

Example:

```
LIB "grobcov.lib";
// EXAMPLE:
if(defined(R)){kill R;}
ring R=(0,x,y,z,t),(x1,y1),lp;
ideal a1=x;
ideal a2=x,y;
ideal a3=x,y,z;
ideal a4=x,y,z,t;
ideal b1=y;
ideal b2=y,z;
ideal b3=y,z,t;
ideal b4=1;
list L1=a1,a2,a3,a4;
list L2=b1,b2,b3,b4;
L1;
↦ [1]:
↦    _[1]=(x)
↦ [2]:
↦    _[1]=(x)
↦    _[2]=(y)
↦ [3]:
```

```
↦    _[1]=(x)
↦    _[2]=(y)
↦    _[3]=(z)
↦ [4]:
↦    _[1]=(x)
↦    _[2]=(y)
↦    _[3]=(z)
↦    _[4]=(t)
L2;
↦ [1]:
↦    _[1]=(y)
↦ [2]:
↦    _[1]=(y)
↦    _[2]=(z)
↦ [3]:
↦    _[1]=(y)
↦    _[2]=(z)
↦    _[3]=(t)
↦ [4]:
↦    _[1]=1
def LL=DifConsLCSets(L1,L2);
LL;
↦ [1]:
↦    [1]:
↦       _[1]=(x)
↦    [2]:
↦       _[1]=(y)
↦       _[2]=(x)
↦ [2]:
↦    [1]:
↦       _[1]=(z)
↦       _[2]=(y)
↦       _[3]=(x)
↦    [2]:
↦       _[1]=(t)
↦       _[2]=(z)
↦       _[3]=(y)
↦       _[4]=(x)
def LLL=ConsLevels(LL);
LLL;
↦ [1]:
↦    _[1]=(x)
↦ [2]:
↦    _[1]=(y)
↦    _[2]=(x)
↦ [3]:
↦    _[1]=(z)
↦    _[2]=(y)
↦    _[3]=(x)
↦ [4]:
↦    _[1]=(t)
↦    _[2]=(z)
↦    _[3]=(y)
↦      _[4]=(x)
↦   [5]:
↦      _[1]=1
def LLLL=Levels(LLL);
LLLL;
↦   [1]:
↦      [1]:
↦         1
↦      [2]:
↦         [1]:
↦            _[1]=(x)
↦         [2]:
↦            _[1]=(y)
↦            _[2]=(x)
↦   [2]:
↦      [1]:
↦         3
↦      [2]:
↦         [1]:
↦            _[1]=(z)
↦            _[2]=(y)
↦            _[3]=(x)
↦         [2]:
↦            _[1]=(t)
↦            _[2]=(z)
↦            _[3]=(y)
↦            _[4]=(x)
```

D.2.5 inout_lib

Library: inout.lib

Purpose: Printing and Manipulating In- and Output

Procedures:

D.2.5.1 allprint

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: allprint(L); L list

Display: prints L[1], L[2], ... if an integer with name ALLprint is defined.
makes "pause", if ALLprint > 0

Return: no return value

Example:

```
LIB "inout.lib";
ring S;
matrix M=matrix(freemodule(2),3,3);
int ALLprint; export ALLprint;
allprint("M =",M);
↦ M =
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,0
kill ALLprint;
```

D.2.5.2 lprint

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: lprint(id[,n]); id poly/ideal/vector/module/matrix, n integer

Return: string of id in a format fitting into lines of size n, such that no monomial gets destroyed, i.e. the new line starts with + or -; (default: n = pagewidth).

Note: id is printed columnwise, each column separated by a blank line; hence lprint(transpose(id)); displays a matrix id in a format which can be used as input.

Example:

```
LIB "inout.lib";
ring r= 0,(x,y,z),ds;
poly f=((x+y)*(x-y)*(x+z)*(y+z)^2);
lprint(f,40);
↦   x3y2-xy4+2x3yz+x2y2z-2xy3z-y4z+x3z2
↦ +2x2yz2-xy2z2-2y3z2+x2z3-y2z3
module m = [f*(x-y)],[0,f*(x-y)];
string s=lprint(m); s;"";
↦   x4y2-x3y3-x2y4+xy5+2x4yz-x3y2z-3x2y3z+xy4z+y5z+x4z2+x3yz2-3x2y2z2-xy3z2
↦ +2y4z2+x3z3-x2yz3-xy2z3+y3z3,
↦   0,
↦ 
↦   0,
↦   x4y2-x3y3-x2y4+xy5+2x4yz-x3y2z-3x2y3z+xy4z+y5z+x4z2+x3yz2-3x2y2z2-xy3z2
↦ +2y4z2+x3z3-x2yz3-xy2z3+y3z3
↦ 
execute("matrix M[2][2]="+s+";");      //use the string s as input
module m1 = transpose(M);               //should be the same as m
print(matrix(m)-matrix(m1));
↦ 0,0,
↦ 0,0
```

D.2.5.3 pmat

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: pmat(M[,n]); M matrix, n integer

Return: A string representing M in array format if it fits into pagewidth; if n is given, only the first n characters of each column are shown (n>1 required), where a truncation of a column is indicated by two dots (\'..\')

Example:

```
LIB "inout.lib";
ring r=0,(x,y,z),ls;
ideal i= x,z+3y,x+y,z;
matrix m[3][3]=i^2;
pmat(m);
↦ x2,     xz+3xy,     xy+x2,
↦ xz,     z2+6yz+9y2, yz+3y2+xz+3xy,
↦ z2+3yz, y2+2xy+x2,  yz+xz
pmat(m,5);
↦ x2,    xz+.., xy+x2,
↦ xz,    z2+.., yz+..,
↦ z2+.., y2+.., yz+xz
```

D.2.5.4 rMacaulay

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: rMacaulay(s[,n]); s string, n integer

Return: A string denoting a file which should be readable by Singular and it should be produced by Macaulay Classic.
If a second argument is present the first
n lines of the file are deleted (which is useful if the file was produced e.g. by the putstd command of Macaulay).

Note: This does not always work with 'cut and paste' since the character \ is treated differently

Example:

```
LIB "inout.lib";
// Assume there exists a file 'Macid' with the following ideal in
// Macaulay format:"
// x[0]3-101/74x[0]2x[1]+7371x[0]x[1]2-13/83x[1]3-x[0]2x[2] \
//     -4/71x[0]x[1]x[2]
// Read this file into Singular and assign it to the string s1 by:
// string s1 = read("Macid");
// This is equivalent to";
string s1 =
"x[0]3-101/74x[0]2x[1]+7371x[0]x[1]2-13/83x[1]3-x[0]2x[2]-4/71x[0]x[1]x[2]";
rMacaulay(s1);
↦ x(0)^3-101/74*x(0)^2*x(1)+7371*x(0)*x(1)^2-13/83*x(1)^3-x(0)^2*x(2)-4/71*\
  x(0)*x(1)*x(2)
// You may wish to assign s1 to a Singular ideal id:
string sid = "ideal id =",rMacaulay(s1),";";
ring r = 0,x(0..3),dp;
execute(sid);
id; "";
↦ id[1]=x(0)^3-101/74*x(0)^2*x(1)+7371*x(0)*x(1)^2-13/83*x(1)^3-x(0)^2*x(2)\
  -4/71*x(0)*x(1)*x(2)
↦
// Now treat a matrix in Macaulay format. Using the execute
// command, this could be assigned to a Singular matrix as above.
string s2 = "
0  0  0  0  0
a3 0  0  0  0
0  b3 0  0  0
0  0  c3 0  0
0  0  0  d3 0
0  0  0  0  e3 ";
rMacaulay(s2);
↦ 0, 0, 0, 0, 0,
↦ a3,0, 0, 0, 0,
↦ 0, b3,0, 0, 0,
↦ 0, 0, c3,0, 0,
↦ 0, 0, 0, d3,0,
↦ 0, 0, 0, 0, e3
```

D.2.5.5 show

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: show(id); id any object of basering or of type ring/qring
show(R,s); R=ring, s=string (s = name of an object belonging to R)

Display: display id/s in a compact format together with some information

Return: no return value

Note: objects of type string, int, intvec, intmat belong to any ring. id may be a ring or a qring. In this case the minimal polynomial is displayed, and, for a qring, also the defining ideal.
id may be of type list but the list must not contain a ring.
show(R,s) does not work inside a procedure!

Example:

```
LIB "inout.lib";
ring r;
show(r);
↦ // ring: (ZZ/32003),(x,y,z),(dp(3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
ideal i=x^3+y^5-6*z^3,xy,x3-y2;
show(i,3);            // introduce 3 space tabs before information
↦    // ideal, 3 generator(s)
↦ y5+x3-6z3,
↦ xy,
↦ x3-y2
vector v=x*gen(1)+y*gen(3);
module m=v,2*v+gen(4);
list L = i,v,m;
show(L);
↦ // list, 3 element(s):
↦ [1]:
↦    // ideal, 3 generator(s)
↦ y5+x3-6z3,
↦ xy,
↦ x3-y2
↦ [2]:
↦    // vector
↦ [x,0,y]
↦ [3]:
↦    // module, 2 generator(s)
↦ [x,0,y]
↦ [2x,0,2y,1]
ring S=(0,T),(a,b,c,d),ws(1,2,3,4);
minpoly = T^2+1;
ideal i=a2+b,c2+T^2*d2; i=std(i);
```

```
qring Q=i;
show(Q);
↦ // ring: (0,T),(a,b,c,d),(ws(1,2,3,4),C);
↦ // minpoly = (T2+1)
↦ // quotient ring from ideal:
↦ _[1]=a2+b
↦ _[2]=c2-d2
↦ // objects belonging to this ring:
map F=r,a2,b^2,3*c3;
show(F);
↦ // i-th variable of preimage ring is mapped to @map[i]
↦ // @map map  from r
↦ @map[1]=a2
↦ @map[2]=b2
↦ @map[3]=3*c3
// Apply 'show' to i (which does not belong to the basering) by typing
// ring r; ideal i=xy,x3-y2; ring Q; show(r,"i");
```

D.2.5.6 showrecursive

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: showrecursive(id,p[,ord]); id any object of basering, p= product of variables and ord=string (any allowed ordstr)

Display: display 'id' in a recursive format as a polynomial in the variables occurring in p with coefficients in the remaining variables. This is done by mapping to a ring with parameters [and ordering 'ord', if a 3rd argument is present (default: ord="dp")] and applying procedure 'show'

Return: no return value

Example:

```
LIB "inout.lib";
ring r=2,(a,b,c,d,x,y),ds;
poly f=y+ax2+bx3+cx2y2+dxy3;
showrecursive(f,x);
↦ // poly, 4 monomial(s)
↦ (b)*x3+(a+cy2)*x2+(dy3)*x+(y)
showrecursive(f,xy,"lp");
↦ // poly, 5 monomial(s)
↦ (b)*x3+(c)*x2y2+(a)*x2+(d)*xy3+y
```

D.2.5.7 split

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: split(s[,n]); s string, n integer

Return: same string, split into lines of length n separated by \ (default: n=pagewidth)

Note: may be used in connection with lprint

Example:

```
LIB "inout.lib";
ring r= 0,(x,y,z),ds;
```

```
poly f = (x+y+z)^4;
split(string(f),50);
↦ x4+4x3y+6x2y2+4xy3+y4+4x3z+12x2yz+12xy2z+4y3z+6x\
↦ 2z2+12xyz2+6y2z2+4xz3+4yz3+z4
split(lprint(f));
↦   x4+4x3y+6x2y2+4xy3+y4+4x3z+12x2yz+12xy2z+4y3z+6x2z2+12xyz2+6y2z2+4xz3+4\
  yz3\
↦ +z4
```

D.2.5.8 tab

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: tab(n); n integer

Return: string of n space tabs

Example:

```
LIB "inout.lib";
for(int n=0; n<=5; n=n+1)
{ tab(5-n)+"*"+tab(n)+"+"+tab(n)+"*"; }
↦      ***
↦     * + *
↦    *  +  *
↦   *   +   *
↦  *    +    *
↦ *     +     *
```

D.2.5.9 pause

Procedure from library `inout.lib` (see Section D.2.5 [inout_lib], page 877).

Usage: pause([prompt]) prompt string

Return: none

Purpose: interrupt the execution of commands, displays prompt or pause and waits for user input

Note: pause is useful in procedures in connection with printlevel to interrupt the computation and to display intermediate results.

Example:

```
LIB "inout.lib";
// can only be shown interactively, try the following commands:
// pause("press <return> to continue");
// pause();
// In the following pocedure TTT, xxx is printed and the execution of
// TTT is stopped until the return-key is pressed, if printlevel>0.
// xxx may be any result of a previous computation or a comment, etc:
//
// proc TTT
// { int pp = printlevel-voice+2;  //pp=0 if printlevel=0 and if TTT is
//    ....                         //not called from another procedure
//    if( pp>0 )
//    {
//       print( xxx );
//       pause("press <return> to continue");
//    }
//      ....
// }
```

See also: Section 5.3.6 [printlevel], page 304; Section 5.1.130 [read], page 250.

D.2.6 modular_lib

Library: modular.lib

Purpose: An abstraction layer for modular techniques

Author: Andreas Steenpass, e-mail: steenpass@mathematik.uni-kl.de

Overview: This library is an abstraction layer for modular techniques which are well-known to speed up many computations and to be easy parallelizable.
The basic idea is to execute some computation modulo several primes and then to lift the result back to characteristic zero via the farey rational map and chinese remaindering. It is thus possible to overcome the often problematic coefficient swell and to run the modular computations in parallel.
In Singular, modular techniques have been quite successfully employed for several applications. A first implementation was done for Groebner bases in Singular's Section D.4.16 [modstd_lib], page 1146, a pioneering work by Stefan Steidel. Since the algorithm is basically the same for all applications, this library aims at preventing library authors from writing the same code over and over again by providing an appropriate abstraction layer. It also offers one-line commands for ordinary Singular users who want to take advantage of modular techniques for their own calculations. Thus modular techniques can be regarded as a parallel skeleton of their own.
The terminology (such as 'pTest' and 'finalTest') follows Singular's Section D.4.16 [modstd_lib], page 1146 and [1].

References:
[1] Nazeran Idrees, Gerhard Pfister, Stefan Steidel: Parallelization of Modular Algorithms. Journal of Symbolic Computation 46, 672-684 (2011). http://arxiv.org/abs/1005.5663

Procedures: See also: Section D.4.3 [assprimeszerodim_lib], page 1013; Section 4.10 [link], page 96; Section D.4.16 [modstd_lib], page 1146; Section D.2.7 [parallel_lib], page 884; Section D.2.13 [tasks_lib], page 958.

D.2.6.1 modular

Procedure from library `modular.lib` (see Section D.2.6 [modular_lib], page 883).

Usage: modular(command, arguments[, primeTest, deleteUnluckyPrimes, pTest, finalTest, pmax), command string, arguments list, primeTest proc, deleteUnluckyPrimes proc, pTest proc, finalTest proc, pmax int

Return: the result of `command` applied to `arguments`, computed using modular methods.

Note: For the general algorithm and the role of the optional arguments primeTest, deleteUnluckyPrimes, pTest, and finalTest, see Section D.4.16.1 [modStd], page 1146 and the reference given in Section D.2.6 [modular_lib], page 883. The default for these arguments is that all tests succeed and that all primes are assumed to be lucky.
The type of the result when `command` is applied to `arguments` must be either `bigint`, `ideal`, `module`, or `matrix`.

The optional argument pmax is an upper bound for the prime numbers to be used for the modular computations. The default is 2147483647 (largest prime which can be represented as an `int` in Singular), or 536870909 (largest prime below 2^29} for baserings with parameters.

Example:

```
LIB "modular.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
modular("std", list(I));
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
```

See also: Section D.4.16.1 [modStd], page 1146.

D.2.7 parallel_lib

Library: parallel.lib

Purpose: An abstraction layer for parallel skeletons

Author: Andreas Steenpass, e-mail: steenpass@mathematik.uni-kl.de

Overview: This library provides implementations of several parallel 'skeletons' (i.e. ways in which parallel tasks rely upon and interact with each other). It is based on the library tasks.lib and aims at both ordinary Singular users as well as authors of Singular libraries.

Procedures: See also: Section D.4.16 [modstd_lib], page 1146; Section D.4.23 [normal_lib], page 1183; Section D.2.11 [resources_lib], page 941; Section D.2.13 [tasks_lib], page 958.

D.2.7.1 parallelWaitN

Procedure from library `parallel.lib` (see Section D.2.7 [parallel_lib], page 884).

Usage: parallelWaitN(commands, arguments, N[, timeout]); commands list, arguments list, N int, timeout int

Return: a list, containing the results of commands[i] applied to arguments[i], i = 1, ..., size(arguments).
The procedure waits for N jobs to finish.
An optional timeout in ms can be provided. Default is 0 which disables the timeout.

Note: The entries of the list commands must be strings. The entries of the list arguments must be lists.
The type of any entry of the returned list whose corresponding task did not finish (due to timeout or error) is "none".
The returned list may contain more than N results if several jobs finished "at the same time". It may contain less than N results in the case of timeout or errors occurring.

Example:

```
LIB "parallel.lib";
ring R = 0, (x,y,z), lp;
ideal I = 3x3y+x3+xy3+y2z2, 2x3z-xy-xz3-y4-z2, 2x2yz-2xy2+xz2-y4;
ideal J = x10+x9y2, x2y7-y8;
list commands = list("std", "std");
list arguments = list(list(I), list(J));
parallelWaitN(commands, arguments, 1);
↦ [2]:
↦    _[1]=y15-y12
↦    _[2]=xy12+y14
↦    _[3]=x2y7-y8
↦    _[4]=x10+x9y2
```

See also: Section D.2.7.3 [parallelWaitAll], page 885; Section D.2.7.2 [parallelWaitFirst], page 885; Section D.2.13 [tasks_lib], page 958.

D.2.7.2 parallelWaitFirst

Procedure from library `parallel.lib` (see Section D.2.7 [parallel_lib], page 884).

Usage: parallelWaitFirst(commands, args[, timeout]); commands list, arguments list, timeout int

Return: a list, containing at least one (if no timeout occurs) of the results of commands[i] applied to arguments[i], i = 1, ..., size(arguments).
The command `parallelWaitFirst(commands, arguments[, timeout])` is synonymous to `parallelWaitN(commands, arguments, 1[, timeout])`. See Section D.2.7.1 [parallelWaitN], page 884 for details on optional arguments and other remarks.

Example:

```
LIB "parallel.lib";
ring R = 0, (x,y,z), lp;
ideal I = 3x3y+x3+xy3+y2z2, 2x3z-xy-xz3-y4-z2, 2x2yz-2xy2+xz2-y4;
ideal J = x10+x9y2, x2y7-y8;
list commands = list("std", "std");
list arguments = list(list(I), list(J));
parallelWaitFirst(commands, arguments);
↦ [2]:
↦    _[1]=y15-y12
↦    _[2]=xy12+y14
↦    _[3]=x2y7-y8
↦    _[4]=x10+x9y2
```

See also: Section D.2.7.3 [parallelWaitAll], page 885; Section D.2.7.1 [parallelWaitN], page 884; Section D.2.13 [tasks_lib], page 958.

D.2.7.3 parallelWaitAll

Procedure from library `parallel.lib` (see Section D.2.7 [parallel_lib], page 884).

Usage: parallelWaitAll(commands, arguments[, timeout]); commands list or string, arguments list, timeout int

Return: a list, containing the results of commands[i] applied to arguments[i], i = 1, ..., size(arguments).
The command `parallelWaitAll(commands, arguments[, timeout])` is synonymous to `@code{`parallelWaitN(commands, arguments, size(arguments)[, timeout])`}`. See Section D.2.7.1 [parallelWaitN], page 884 for details on optional arguments and other remarks.

Note: As a shortcut, `commands` can be a string. This is synonymous to providing a list of `size(arguments)` copies of this string.

Example:

```
LIB "parallel.lib";
ring R = 0, (x,y,z), dp;
ideal I1 = z8+z6+4z5+4z3+4z2+4, -z2+y;
ideal I2 = x9y2+x10, x2y7-y8;
ideal I3 = x3-2xy, x2y-2y2+x;
string command = "std";
list arguments = list(list(I1), list(I2), list(I3));
parallelWaitAll(command, arguments);
↦ [1]:
↦    _[1]=z2-y
↦    _[2]=y4+y3+4y2z+4yz+4y+4
↦ [2]:
↦    _[1]=x2y7-y8
↦    _[2]=x9y2+x10
↦    _[3]=x12y+xy11
↦    _[4]=x13-xy12
↦    _[5]=y14+xy12
↦    _[6]=xy13+y12
↦ [3]:
↦    _[1]=2y2-x
↦    _[2]=xy
↦    _[3]=x2
```

See also: Section D.2.7.2 [parallelWaitFirst], page 885; Section D.2.7.1 [parallelWaitN], page 884; Section D.2.13 [tasks_lib], page 958.

D.2.7.4 parallelTestAND

Procedure from library `parallel.lib` (see Section D.2.7 [parallel_lib], page 884).

Usage: parallelTestAND(commands, arguments[, timeout]); commands list or string, arguments list, timeout int

Return: 1, if commands[i] applied to arguments[i] is not equal to zero for all i = 1, ..., size(arguments);
0, otherwise.
An optional timeout in ms can be provided. Default is 0 which disables the timeout.
In case of timeout, -1 is returned.

Note: The entries of the list commands must be strings. The entries of the list arguments must be lists.
commands[i] applied to arguments[i] must evaluate to an integer for i = 1, ..., size(arguments).
As a shortcut, `commands` can be a string. This is synonymous to providing a list of `size(arguments)` copies of this string.

Example:

```
LIB "parallel.lib";
ring R = 0, (x,y,z), dp;
ideal I = x, y, z;
intvec v = 0:3;
```

```
list l = list(I, v);
module m1 = x*gen(1);
module m2;
string command = "size";
list arguments1 = list(list(I), list(v), list(l), list(m1));
list arguments2 = list(list(I), list(v), list(l), list(m2));
// test if all the arguments have non-zero size
parallelTestAND(command, arguments1);
↦ 1
parallelTestAND(command, arguments2);
↦ 0
```

See also: Section D.2.7.5 [parallelTestOR], page 887; Section D.2.13 [tasks_lib], page 958.

D.2.7.5 parallelTestOR

Procedure from library `parallel.lib` (see Section D.2.7 [parallel_lib], page 884).

Usage: parallelTestOR(commands, arguments[, timeout]); commands list or string, arguments list, timeout int

Return: 1, if commands[i] applied to arguments[i] is not equal to zero for any i = 1, ..., size(arguments);
0, otherwise.
An optional timeout in ms can be provided. Default is 0 which disables the timeout. In case of timeout, -1 is returned.

Note: The entries of the list commands must be strings. The entries of the list arguments must be lists.
commands[i] applied to arguments[i] must evaluate to an integer for i = 1, ..., size(arguments).
As a shortcut, `commands` can be a string. This is synonymous to providing a list of `size(arguments)` copies of this string.

Example:

```
LIB "parallel.lib";
ring R = 0, (x,y,z), dp;
ideal I;
string s;
list l;
module m1 = x*gen(1);
module m2;
string command = "size";
list arguments1 = list(list(I), list(s), list(l), list(m1));
list arguments2 = list(list(I), list(s), list(l), list(m2));
// test if any of the arguments has non-zero size
parallelTestOR(command, arguments1);
↦ 1
parallelTestOR(command, arguments2);
↦ 0
```

See also: Section D.2.7.4 [parallelTestAND], page 886; Section D.2.13 [tasks_lib], page 958.

D.2.8 polylib_lib

Library: polylib.lib

Purpose: Procedures for Manipulating Polys, Ideals, Modules

Authors: O. Bachmann, G.-M. Greuel, A. Fruehbis

Procedures:

D.2.8.1 cyclic

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: cyclic(n); n integer

Return: ideal of cyclic n-roots from 1-st n variables of basering

Example:

```
LIB "polylib.lib";
ring r=0,(u,v,w,x,y,z),lp;
cyclic(nvars(basering));
↦ _[1]=u+v+w+x+y+z
↦ _[2]=uv+uz+vw+wx+xy+yz
↦ _[3]=uvw+uvz+uyz+vwx+wxy+xyz
↦ _[4]=uvwx+uvwz+uvyz+uxyz+vwxy+wxyz
↦ _[5]=uvwxy+uvwxz+uvwyz+uvxyz+uwxyz+vwxyz
↦ _[6]=uvwxyz-1
homog(cyclic(5),z);
↦ _[1]=u+v+w+x+y
↦ _[2]=uv+uy+vw+wx+xy
↦ _[3]=uvw+uvy+uxy+vwx+wxy
↦ _[4]=uvwx+uvwy+uvxy+uwxy+vwxy
↦ _[5]=uvwxy-z5
```

D.2.8.2 elemSymmId

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Return: ideal of elementary symmetric polynomials for 1-st n
variables of basering

Example:

```
LIB "polylib.lib";
ring R = 0, (v,w,x,y,z), lp;
elemSymmId(3);
↦ _[1]=v+w+x
↦ _[2]=vw+vx+wx
↦ _[3]=vwx
elemSymmId(nvars(basering));
↦ _[1]=v+w+x+y+z
↦ _[2]=vw+vx+vy+vz+wx+wy+wz+xy+xz+yz
↦ _[3]=vwx+vwy+vwz+vxy+vxz+vyz+wxy+wxz+wyz+xyz
↦ _[4]=vwxy+vwxz+vwyz+vxyz+wxyz
↦ _[5]=vwxyz
```

D.2.8.3 katsura

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: katsura([n]); n integer

Return: katsura(n) : n-th katsura ideal of
(1) newly created and set ring (32003, x(0..n), dp), if nvars(basering) < n
(2) basering, if nvars(basering) >= n
katsura() : katsura ideal of basering

Example:

```
LIB "polylib.lib";
ring r; basering;
↦ // coefficients: ZZ/32003
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
katsura();
↦ _[1]=x+2y+2z-1
↦ _[2]=x2+2y2+2z2-x
↦ _[3]=2xy+2yz-y
katsura(4); basering;
↦ _[1]=x(0)+2*x(1)+2*x(2)+2*x(3)-1
↦ _[2]=x(0)^2+2*x(1)^2+2*x(2)^2+2*x(3)^2-x(0)
↦ _[3]=2*x(0)*x(1)+2*x(1)*x(2)+2*x(2)*x(3)-x(1)
↦ _[4]=x(1)^2+2*x(0)*x(2)+2*x(1)*x(3)-x(2)
↦ // coefficients: ZZ/32003
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x(0) x(1) x(2) x(3) x(4)
↦ //        block   2 : ordering C
```

D.2.8.4 freerank

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: freerank(M[,any]); M=poly/ideal/vector/module/matrix

Compute: rank of module presented by M in case it is free.
By definition this is vdim(coker(M)/m*coker(M)) if coker(M) is free, where m is the maximal ideal of the variables of the basering and M is considered to be a matrix.
(the 0-module is free of rank 0)

Return: rank of coker(M) if coker(M) is free and -1 else;
in case of a second argument return a list:
L[1] = rank of coker(M) or -1
L[2] = minbase(M)

Note: freerank(syz(M)); computes the rank of M if M is free (and -1 else)

Example:

```
LIB "polylib.lib";
ring r;
ideal i=x;
module M=[x,0,1],[-x,0,-1];
freerank(M);          // should be 2, coker(M) is not free
↦ 2
freerank(syz (M),"");
↦ [1]:
```

```
↦    1
↦ [2]:
↦    _[1]=gen(2)+gen(1)
// [1] should be 1, coker(syz(M))=M is free of rank 1
// [2] should be gen(2)+gen(1) (minimal relation of M)
freerank(i);
↦ -1
freerank(syz(i));     // should be 1, coker(syz(i))=i is free of rank 1
↦ 1
```

D.2.8.5 is_zero

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: is_zero(M[,any]); M=poly/ideal/vector/module/matrix

Return: integer, 1 if coker(M)=0 resp. 0 if coker(M)!=0, where M is considered as matrix.
If a second argument is given, return a list:
L[1] = 1 if coker(M)=0 resp. 0 if coker(M)!=0
L[2] = dim(M)

Example:

```
LIB "polylib.lib";
ring r;
module m = [x],[y],[1,z];
is_zero(m,1);
↦ [1]:
↦    0
↦ [2]:
↦    2
qring q = std(ideal(x2+y3+z2));
ideal j = x2+y3+z2-37;
is_zero(j);
↦ 1
```

D.2.8.6 lcm

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: lcm(p[,q]); p int/intvec q a list of integers or
p poly/ideal q a list of polynomials

Return: the least common multiple of p and q:
- of type int if p is an int/intvec
- of type poly if p is a poly/ideal

Example:

```
LIB "polylib.lib";
ring  r = 0,(x,y,z),lp;
poly  p = (x+y)*(y+z);
poly  q = (z4+2)*(y+z);
lcm(p,q);
↦ xyz4+2xy+xz5+2xz+y2z4+2y2+yz5+2yz
ideal i=p,q,y+z;
lcm(i,p);
```

```
↦ xyz4+2xy+xz5+2xz+y2z4+2y2+yz5+2yz
lcm(2,3,6);
↦ 6
lcm(2..6);
↦ 60
```

D.2.8.7 maxcoef

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: maxcoef(f); f poly/ideal/vector/module/matrix

Return: maximal length of coefficient of f of type int (by measuring the length of the string of each coefficient)

Example:

```
LIB "polylib.lib";
ring r= 0,(x,y,z),ds;
poly g = 345x2-1234567890y+7/4z;
maxcoef(g);
↦ 10
ideal i = g,10/1234567890;
maxcoef(i);
↦ 11
// since i[2]=1/123456789
```

D.2.8.8 maxdeg

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: maxdeg(id); id poly/ideal/vector/module/matrix

Return: int/intmat, each component equals maximal degree of monomials in the corresponding component of id, independent of ring ordering (maxdeg of each var is 1).
Of type int, if id is of type poly; of type intmat otherwise

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),wp(1,2,3);
poly f = x+y2+z3;
deg(f);             //deg; returns weighted degree (in case of 1 block)!
↦ 9
maxdeg(f);
↦ 3
matrix m[2][2]=f+x10,1,0,f^2;
maxdeg(m);
↦ 10,0,
↦ -1,6
```

See also: Section D.2.8.9 [maxdeg1], page 891.

D.2.8.9 maxdeg1

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: maxdeg1(id[,v]); id=poly/ideal/vector/module/matrix, v=intvec

Return: integer, maximal [weighted] degree of monomials of id independent of ring ordering, maxdeg1 of i-th variable is v[i] (default: v=1..1).

Note: This proc returns one integer while maxdeg returns, in general, a matrix of integers. For one polynomial and if no intvec v is given maxdeg is faster

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),wp(1,2,3);
poly f = x+y2+z3;
deg(f);            //deg returns weighted degree (in case of 1 block)!
↦ 9
maxdeg1(f);
↦ 3
intvec v = ringweights(r);
maxdeg1(f,v);                        //weighted maximal degree
↦ 9
matrix m[2][2]=f+x10,1,0,f^2;
maxdeg1(m,v);                        //absolute weighted maximal degree
↦ 18
```

D.2.8.10 mindeg

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: mindeg(id); id poly/ideal/vector/module/matrix

Return: minimal degree/s of monomials of id, independent of ring ordering (mindeg of each variable is 1) of type int if id of type poly, else of type intmat.

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),ls;
poly f = x5+y2+z3;
ord(f);                  // ord returns weighted order of leading term!
↦ 3
mindeg(f);               // computes minimal degree
↦ 2
matrix m[2][2]=x10,1,0,f^2;
mindeg(m);               // computes matrix of minimum degrees
↦ 10,0,
↦ -1,4
```

See also: Section D.2.8.11 [mindeg1], page 892.

D.2.8.11 mindeg1

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: mindeg1(id[,v]); id=poly/ideal/vector/module/matrix, v=intvec

Return: integer, minimal [weighted] degree of monomials of id independent of ring ordering, mindeg1 of i-th variable is v[i] (default v=1..1).

Note: This proc returns one integer while mindeg returns, in general, a matrix of integers. For one polynomial and if no intvec v is given mindeg is faster.

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),ls;
poly f = x5+y2+z3;
ord(f);                  // ord returns weighted order of leading term!
↦ 3
intvec v = 1,-3,2;
mindeg1(f,v);            // computes minimal weighted degree
↦ -6
matrix m[2][2]=x10,1,0,f^2;
mindeg1(m,1..3);         // computes absolute minimum of weighted degrees
↦ -1
```

D.2.8.12 normalize

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: normalize(id); id=poly/vector/ideal/module

Return: object of same type
each element is normalized with leading coefficient equal to 1

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),ls;
poly f = 2x5+3y2+4z3;
normalize(f);
↦ z3+3/4y2+1/2x5
module m=[9xy,0,3z3],[4z,6y,2x];
normalize(m);
↦ _[1]=z3*gen(3)+3xy*gen(1)
↦ _[2]=z*gen(1)+3/2y*gen(2)+1/2x*gen(3)
ring s = 0,(x,y,z),(c,ls);
module m=[9xy,0,3z3],[4z,6y,2x];
normalize(m);
↦ _[1]=[xy,0,1/3z3]
↦ _[2]=[z,3/2y,1/2x]
```

D.2.8.13 rad_con

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: rad_con(g,I); g polynomial, I ideal

Return: 1 (TRUE) (type int) if g is contained in the radical of I
0 (FALSE) (type int) otherwise

Example:

```
LIB "polylib.lib";
ring R=0,(x,y,z),dp;
ideal I=x2+y2,z2;
poly f=x4+y4;
rad_con(f,I);
↦ 0
ideal J=x2+y2,z2,x4+y4;
poly g=z;
rad_con(g,I);
↦ 1
```

D.2.8.14 content

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: content(f); f polynomial/vector

Return: number, the content (greatest common factor of coefficients) of the polynomial/vector f

Example:

```
LIB "polylib.lib";
ring r=0,(x,y,z),(c,lp);
content(3x2+18xy-27xyz);
↦ 3
vector v=[3x2+18xy-27xyz,15x2+12y4,3];
content(v);
↦ 3
```

See also: Section 5.1.9 [cleardenom], page 163.

D.2.8.15 mod2id

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: mod2id(M,vpos); M matrix, vpos intvec

Assume: vpos is an integer vector such that gen(i) corresponds to var(vpos[i]).
The basering contains variables var(vpos[i]) which do not occur in M.

Return: ideal I in which each gen(i) from the module is replaced by var(vpos[i]) and all monomials var(vpos[i])*var(vpos[j]) have been added to the generating set of I.

Note: This procedure should be used in the following situation: one wants to pass to a ring with new variables, say e(1),..,e(s), which correspond to the components gen(1),..,gen(s) of the module M such that e(i)*e(j)=0 for all i,j.
The new ring should already exist and be the current ring

Example:

```
LIB "polylib.lib";
ring r=0,(e(1),e(2),x,y,z),(dp(2),ds(3));
module mo=x*gen(1)+y*gen(2);
intvec iv=2,1;
mod2id(mo,iv);
↦ _[1]=e(2)^2
↦ _[2]=e(1)*e(2)
↦ _[3]=e(1)^2
↦ _[4]=e(1)*y+e(2)*x
```

D.2.8.16 id2mod

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: id2mod(I,vpos); I ideal, vpos intvec

Return: module corresponding to the ideal by replacing var(vpos[i]) by gen(i) and omitting all generators var(vpos[i])*var(vpos[j])

Note: * This procedure only makes sense if the ideal contains all var(vpos[i])*var(vpos[j]) as monomial generators and all other generators of I are linear combinations of the var(vpos[i]) over the ring in the other variables.
* This is the inverse procedure to mod2id and should be applied only to ideals created by mod2id using the same intvec vpos (possibly after a standard basis computation)

Example:

```
LIB "polylib.lib";
ring r=0,(e(1),e(2),x,y,z),(dp(2),ds(3));
ideal i=e(2)^2,e(1)*e(2),e(1)^2,e(1)*y+e(2)*x;
intvec iv=2,1;
id2mod(i,iv);
↦ _[1]=x*gen(1)+y*gen(2)
```

D.2.8.17 substitute

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: - case 1: typeof(#[1])==poly:
substitute (I,v,f[,v1,f1,v2,f2,...]); I object of basering which can be mapped, v,v1,v2,.. ring variables, f,f1,f2,... poly
- case 2: typeof(#[1])==ideal: substitute (I,v,f); I object of basering which can be mapped, v ideal of ring variables, f ideal

Return: object of same type as I,
- case 1: ring variable v,v1,v2,... substituted by polynomials f,f1,f2,..., in this order
- case 2: ring variables in v substituted by polynomials in f: v[i] is substituted by f[i], i=1,...,i=min(size(v),ncols(f))

Note: this procedure extends the built-in command subst via maps

Example:

```
LIB "polylib.lib";
ring r = 0,(b,c,t),dp;
ideal I = -bc+4b2c2t,bc2t-5b2c;
substitute(I,c,b+c,t,0,b,b-1);
↦ _[1]=-b2-bc+2b+c-1
↦ _[2]=-5b3-5b2c+15b2+10bc-15b-5c+5
ideal v = c,t,b;
ideal f = b+c,0,b-1;
substitute(I,v,f);
↦ _[1]=-b2-bc+b+c
↦ _[2]=-5b3-5b2c+10b2+10bc-5b-5c
```

D.2.8.18 subrInterred

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: subrInterred(mon,sm,iv);
sm: ideal in a ring r with n + s variables,
e.g. x_1,..,x_n and t_1,..,t_s
mon: ideal with monomial generators (not divisible by any of the t_i) such that sm is contained in the module k[t_1,..,t_s]*mon[1]+..+k[t_1,..,t_s]*mon[size(mon)]
iv: intvec listing the variables which are supposed to be used as x_i

Return: list l:
l[1]=the monomials from mon in the order used
l[2]=their coefficients after interreduction
l[3]=l[1]*l[2]

Purpose: Do interred only w.r.t. a subset of variables.
The procedure returns an interreduced system of generators of sm considered as a k[t_1,..,t_s]-submodule of the free module k[t_1,..,t_s]*mon[1]+..+k[t_1,..,t_s]*mon[size(mon)]).

Example:

```
LIB "polylib.lib";
ring r=0,(x,y,z),dp;
ideal i=x^2+x*y^2,x*y+x^2*y,z;
ideal j=x^2+x*y^2,x*y,z;
ideal mon=x^2,z,x*y;
intvec iv=1,3;
subrInterred(mon,i,iv);
↦ [1]:
↦    _[1,1]=z
↦    _[1,2]=xy
↦    _[1,3]=x2
↦ [2]:
↦    _[1]=gen(1)
↦    _[2]=y2*gen(2)-gen(2)
↦    _[3]=y*gen(2)+gen(3)
↦ [3]:
↦    _[1,1]=z
↦    _[1,2]=xy3-xy
↦    _[1,3]=xy2+x2
subrInterred(mon,j,iv);
↦ [1]:
↦    _[1,1]=z
↦    _[1,2]=xy
↦    _[1,3]=x2
↦ [2]:
↦    _[1]=gen(1)
↦    _[2]=gen(2)
↦    _[3]=gen(3)
↦ [3]:
↦    _[1,1]=z
↦    _[1,2]=xy
↦    _[1,3]=x2
```

D.2.8.19 newtonDiag

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: newtonDiag(f); f a poly

Return: intmat

Purpose: compute the Newton diagram of f

Note: each row is the exponent of a monomial of f

Example:

```
LIB "polylib.lib";
ring r = 0,(x,y,z),lp;
poly f = x2y+3xz-5y+3;
newtonDiag(f);
↦ 2,1,0,
↦ 1,0,1,
↦ 0,1,0,
↦ 0,0,0
```

D.2.8.20 hilbPoly

Procedure from library `polylib.lib` (see Section D.2.8 [polylib_lib], page 887).

Usage: hilbPoly(I); I a homogeneous ideal

Return: the Hilbert polynomial of basering/I as an intvec v=v_0,...,v_r such that the Hilbert polynomial is (v_0+v_1*t+...v_r*t^r)/r!

Example:

```
LIB "polylib.lib";
ring r = 0,(b,c,t,h),dp;
ideal I=
bct-t2h+2th2+h3,
bt3-ct3-t4+b2th+c2th-2bt2h+2ct2h+2t3h-bch2-2bth2+2cth2+2th3,
b2c2+bt2h-ct2h-t3h+b2h2+2bch2+c2h2-2bth2+2cth2+t2h2-2bh3+2ch3+2th3+3h4,
c2t3+ct4-c3th-2c2t2h-2ct3h-t4h+bc2h2-2c2th2-bt2h2+4t3h2+2bth3-2cth3-t2h3
+bh4-6th4-2h5;
hilbPoly(I);
↦ -11,10
```

D.2.9 redcgs_lib

Library: redcgs.lib

Purpose: Reduced Comprehensive Groebner Systems.

Overview: Comprehensive Groebner Systems. Canonical Forms.
The library contains Monte's algorithms to compute disjoint, reduced Comprehensive Groebner Systems (CGS). A CGS is a set of pairs of (segment,basis). The segments S_i are subsets of the parameter space, and the bases B_i are sets of polynomials specializing to Groebner bases of the specialized ideal for every point in S_i.

The purpose of the routines in this library is to obtain CGS with better properties, namely disjoint segments forming a partition of the parameter space and reduced bases. Reduced bases are sets of polynomials that specialize to the reduced Groebner basis of the specialized ideal preserving the leading power products (lpp). The lpp characterize the type of solution in each segment.

A further objective is to summarize as much as possible the segments with the same lpp into a single segment, and if possible to obtain a final result that is canonical, i.e. independent of the algorithm and only attached to the given ideal.

There are three fundamental routines in the library: mrcgs, rcgs and crcgs. mrcgs (Minimal Reduced CGS) is an algorithm that packs so much as it is able to do (using algorithms adhoc) the segments with the same lpp, obtaining the minimal number of segments. The hypothesis is that the result is also canonical, but for the moment there is no proof of the uniqueness of this minimal packing. Moreover, the segments that are obtained are not locally closed, i.e. there are not difference of two varieties.

On the other side, Michael Wibmer has proved that for homogeneous ideals, all the segments with reduced bases having the same lpp admit a unique basis specializing well. For this purpose it is necessary to extend the description of the elements of the bases to functions, forming sheaves of polynomials instead of simple polynomials, so that the polynomials in a sheaf either preserve the lpp of the corresponding polynomial of the specialized Groebner basis (and then it specializes well) or it specializes to 0. Moreover, in a sheaf, for every point in the corresponding segment, at least one of the polynomials specializes well. specializes well. And moreover Wibmer's Theorem ensures that the packed segments are locally closed, that is can be described as the difference of two varieties.

Using Wibmer's Theorem we proved that an affine ideal can be homogenized, than discussed by mrcgs and finally de-homogenized. The bases so obtained can be reduced and specialize well in the segment. If the theoretic objective is reached, and all the segments of the homogenized ideal have been packed, locally closed segments will be obtained.

If we only homogenize the given basis of the ideal, then we cannot ensure the canonicity of the partition obtained, because there are many different bases of the given ideal that can be homogenized, and the homogenized ideals are not identical. This corresponds to the algorithm rcgs and is recommended as the most practical routine. It provides locally closed segments and is usually faster than mrcgs and crcgs. But the given partition is not always canonical.

Finally it is possible to homogenize the whole affine ideal, and then the packing algorithm will provide canonical segments by dehomogenizing. This corresponds to crcgs routine. It provides the best description of the segments and bases. In contrast crcgs algorithm is usually much more time consuming and it will not always finish in a reasonable time. Moreover it will contain more segments than mrcgs and possibly also more than rcgs.

But the actual algorithms in the library to pack segments have some lacks. They are not theoretically always able to pack the segments that we know that can be packed. Nevertheless, thanks to Wibmer's Theorem, the algorithms rcgs and crcgs are able to detect if the objective has not been reached, and if so, to give a Warning. The warning does not invalidate the output, but it only recognizes that the theoretical objective is not completely reached by the actual computing methods and that some segments that can be packed have not been packed with a single basis.

The routine buildtree is the first algorithm used in all the previous methods providing a first disjoint CGS, and can be used if none of the three fundamental algorithms of the library finishes in a reasonable time.

There are also routines to visualize better the output of the previous algorithms:
finalcases can be applied to the list provided by buildtree to obtain the CGS. The list provided by buildtree contains the whole discussion, and finalcases extracts the CGS.
The output of buildtree can also be transformed into a file using buildtreetoMaple routine that can be read in Maple. Using Monte's dpgb library in Maple the output can be plotted (with the routine tplot). To plot the output of mrcgs, rcgs or crcgs in Maple, the library also provides the routine cantreetoMaple. The file written using it and read in Maple can then be plotted with the command plotcantree and printed with printcantree from the Monte's dpgb library in Maple. The output of mrcgs,

rcgs and crcgs is given in form of tree using prime ideals in a canonical form that is described in the papers. Nevertheless this canonical form is somewhat uncomfortable to be interpreted. When the segments are all locally closed (and this is always the case for rcgs and crcgs) the routine cantodiffcgs transforms the output into a simpler form having only one list element for each segment and providing the two varieties whose difference represent the segment also in a canonical form.

Authors: Antonio Montes , Hans Schoenemann.

Overview: see "Minimal Reduced Comprehensive Groebner Systems" by Antonio Montes. (http://www-ma2.upc.edu/~montes/).

Notations: All given and determined polynomials and ideals are in the
basering K[a][x]; (a=parameters, x=variables)
After defining the ring and calling setglobalrings(); the rings
@R (K[a][x]),
@P (K[a]),
@RP (K[x,a]) are defined globally
They are used internally and can also be used by the user.
The fundamental routines are: buildtree, mrcgs, rcgs and crcgs

Procedures: See also: Section D.2.2 [compregb_lib], page 797.

D.2.9.1 setglobalrings

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: setglobalrings();
No arguments

Return: After its call the rings @R=K[a][x], @P=K[a], @RP=K[x,a] are defined as global variables.

Note: It is called by the fundamental routines of the library. The user does not need to call it, except when none of
the fundamental routines have been called and some
other routines of the library are used.
The basering R, must be of the form K[a][x], a=parameters, x=variables, and should be defined previously.

Example:

```
LIB "redcgs.lib";
ring R=(0,a,b),(x,y,z),dp;
setglobalrings();
@R;
↦ // coefficients: QQ(a, b)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
@P;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering lp
↦ //                  : names    a b
↦ //        block   2 : ordering C
```

```
@RP;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering lp
↦ //                  : names    a b
↦ //        block   3 : ordering C
```

D.2.9.2 memberpos

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: memberpos(f,J);
(f,J) expected (polynomial,ideal)
or (int,list(int))
or (int,intvec)
or (intvec,list(intvec))
or (list(int),list(list(int)))
or (ideal,list(ideal))
or (list(intvec), list(list(intvec))).
The ring can be `@R` or `@P` or `@RP` or any other.

Return: The list (t,pos) t int; pos int;
t is 1 if f belongs to J and 0 if not.
pos gives the position in J (or 0 if f does not belong).

Example:

```
LIB "redcgs.lib";
list L=(7,4,5,1,1,4,9);
memberpos(1,L);
↦ [1]:
↦    1
↦ [2]:
↦    4
```

D.2.9.3 subset

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: subset(J,K);
(J,K) expected (ideal,ideal)
or (list, list)

Return: 1 if all the elements of J are in K, 0 if not.

Example:

```
LIB "redcgs.lib";
list J=list(7,3,2);
list K=list(1,2,3,5,7,8);
subset(J,K);
↦ 1
```

D.2.9.4 pdivi2

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: pdivi2(f,F);
poly f: the polynomial to be divided
ideal F: the divisor ideal

Return: A list (poly r, ideal q, poly m). r is the remainder of the pseudodivision, q is the ideal of quotients, and m is the factor by which f is to be multiplied.

Note: Pseudodivision of a polynomial f by an ideal F in @R. Returns a list (r,q,m) such that m*f=r+sum(q.G).

Example:

```
LIB "redcgs.lib";
ring R=(0,a,b,c),(x,y),dp;
setglobalrings();
poly f=(ab-ac)*xy+(ab)*x+(5c);
ideal F=ax+b,cy+a;
def r=pdivi2(f,F);
r;
↦ [1]:
↦    (ab2-abc-b2c+5c2)
↦ [2]:
↦    _[1]=(bc-c2)*y+(bc)
↦    _[2]=(-b2+bc)
↦ [3]:
↦    (c)
r[3]*f-(r[2][1]*F[1]+r[2][2]*F[2])-r[1];
↦ 0
```

D.2.9.5 facvar

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: facvar(J);
J: an ideal in the parameters

Return: all the free-square factors of the elements
of ideal J (non repeated). Integer factors are ignored, even 0 is ignored. It can be called from ideal @R, but the given ideal J must only contain poynomials in the parameters.

Note: Operates in the ring @P, and the ideal J must contain only polynomials in the parameters, but can be called from ring @R.

Example:

```
LIB "redcgs.lib";
ring R=(0,a,b,c),(x,y,z),dp;
setglobalrings();
ideal J=a2-b2,a2-2ab+b2,abc-bc;
facvar(J);
↦ _[1]=(a-b)
↦ _[2]=(a+b)
↦ _[3]=(a-1)
↦ _[4]=(b)
↦ _[5]=(c)
```

D.2.9.6 redspec

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: redspec(N,W);
N: null conditions ideal
W: set of non-null polynomials (ideal)

Return: a list (N1,W1,L1) containing a red-specification of the segment (N,W). N1 is the radical reduced ideal characterizing the segment. V(N1) is the Zarisky closure of the segment (N,W).
The segment S=V(N1) \ V(h), where h=prod(w in W1)
N1 is uniquely determined and no prime component of N1 contains none of the polynomials in W1. The polynomials in W1 are prime and reduced w.r.t. N1, and are considered non-null on the segment. L1 contains the list of prime components of N1.

Note: can be called from ring @R but it works in ring @P.

Example:

```
LIB "redcgs.lib";
ring r=(0,a,b,c),(x,y),dp;
setglobalrings();
ideal N=(ab-c)*(a-b),(a-bc)*(a-b);
ideal W=a^2-b^2,bc;
redspec(N,W);
↦ [1]:
↦    _[1]=(b2-1)
↦    _[2]=(a-bc)
↦ [2]:
↦    _[1]=(b)
↦    _[2]=(c-1)
↦    _[3]=(c+1)
↦    _[4]=(c)
↦ [3]:
↦    [1]:
↦       _[1]=(b+1)
↦       _[2]=(a+c)
↦    [2]:
↦       _[1]=(b-1)
↦       _[2]=(a-c)
```

D.2.9.7 pnormalform

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: pnormalform(f,N,W);
f: the polynomial to be reduced modulo N,W (in parameters and variables)
N: the null conditions ideal
W: the non-null conditions (set of irreducible polynomials, ideal)

Return: a reduced polynomial g of f, whose coefficients are reduced modulo N and having no factor in W.

Note: Should be called from ring @R. Ideals N and W must be polynomials in the parameters forming a red-specification (see definition) the papers).

Example:

```
LIB "redcgs.lib";
ring R=(0,a,b,c),(x,y),dp;
setglobalrings();
poly f=(b^2-1)*x^3*y+(c^2-1)*x*y^2+(c^2*b-b)*x+(a-bc)*y;
ideal N=(ab-c)*(a-b),(a-bc)*(a-b);
ideal W=a^2-b^2,bc;
def r=redspec(N,W);
pnormalform(f,r[1],r[2]);
↦ xy2+(b)*x
```

D.2.9.8 buildtree

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: buildtree(F);
F: ideal in K[a][x] (parameters and variables) to be discussed

Return: Returns a list T describing a dichotomic discussion tree, whose content is the first discussion of the ideal F of K[a][x]. The first element of the list is the root, and contains
[1] label: intvec(-1)
[2] number of children : int
[3] the ideal F
[4], [5], [6] the red-spec of the null and non-null conditions given (as option). ideal (0), ideal (0), list(ideal(0)) if no optional conditions are given.
[7] the set of lpp of ideal F
[8] condition that was taken to reach the vertex
(poly 1, for the root).
The remaining elements of the list represent vertices of the tree: with the same structure:
[1] label: intvec (1,0,0,1,...) gives its position in the tree: first branch condition is taken non-null, second null,... [2] number of children (0 if it is a terminal vertex) [3] the specialized ideal with the previous assumed conditions to reach the vertex
[4],[5],[6] the red-spec of the previous assumed conditions to reach the vertex
[7] the set of lpp of the specialized ideal at this stage [8] condition that was taken to reach the vertex from the father's vertex (that was taken non-null if the last
integer in the label is 1, and null if it is 0)
The terminal vertices form a disjoint partition of the parameter space whose bases specialize to the reduced Groebner basis of the specialized ideal on each point of the segment and preserve the lpp. So they form a disjoint reduced CGS.

Note: The basering R, must be of the form K[a][x], a=parameters, x=variables, and should be defined previously. The ideal must be defined on R.
The disjoint and reduced CGS built by buildtree can be obtained from the output of buildtree by calling finalcases(T); this selects the terminal vertices.
The content of buildtree can be written in a file that is readable by Maple in order to plot its content using buildtreetoMaple; The file written by buildtreetoMaple when read in a Maple worksheet can be plotted using the dbgb routine tplot;

Example:

```
LIB "redcgs.lib";
ring R=(0,a1,a2,a3,a4),(x1,x2,x3,x4),dp;
ideal F=x4-a4+a2,
x1+x2+x3+x4-a1-a3-a4,
x1*x3*x4-a1*a3*a4,
x1*x3+x1*x4+x2*x3+x3*x4-a1*a4-a1*a3-a3*a4;
def T=buildtree(F);
finalcases(T);
↦ [1]:
↦    [1]:
↦       0,0
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x4
↦       _[2]=-x1-x2-x3+(a1+a3+a4)
↦       _[3]=x3^2+(-a1-a3-a4)*x3+(a1*a3+a1*a4+a3*a4)
↦    [4]:
↦       _[1]=(a2-a4)
↦       _[2]=(a1*a3*a4)
↦    [5]:
↦       _[1]=0
↦    [6]:
↦       [1]:
↦          _[1]=(a4)
↦          _[2]=(a2)
↦       [2]:
↦          _[1]=(a3)
↦          _[2]=(a2-a4)
↦       [3]:
↦          _[1]=(a2-a4)
↦          _[2]=(a1)
↦    [7]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦    [8]:
↦       1
↦ [2]:
↦    [1]:
↦       0,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=(a2-a4)
↦    [5]:
↦       _[1]=(a1)
↦       _[2]=(a3)
↦       _[3]=(a4)
↦    [6]:
↦       [1]:
↦          _[1]=(a2-a4)
↦    [7]:
↦       _[1]=1
↦     [8]:
↦        1
↦  [3]:
↦     [1]:
↦        1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=x4+(a2-a4)
↦        _[2]=-x1-x2-x3+(a1+a2+a3)
↦        _[3]=x3^2+(-a2+a4)*x2+(-a1-a2-a3)*x3+(a1*a2+a1*a3+a2^2+a2*a3-a2*a4)
↦        _[4]=(a2-a4)*x2*x3+(a2^2-2*a2*a4+a4^2)*x2+(-a1*a2^2-a1*a2*a3+a1*a2*\
   a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)
↦        _[5]=(a2^2-2*a2*a4+a4^2)*x2^2+(-2*a1*a2^2-a1*a2*a3+3*a1*a2*a4+a1*a3\
   *a4-a1*a4^2-3*a2^3-2*a2^2*a3+7*a2^2*a4+3*a2*a3*a4-5*a2*a4^2-a3*a4^2+a4^3)\
   *x2+(-a1*a2^2-a1*a2*a3+a1*a2*a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)*\
   x3+(a1^2*a2^2+a1^2*a2*a3-a1^2*a2*a4+3*a1*a2^3+4*a1*a2^2*a3-5*a1*a2^2*a4+a\
   1*a2*a3^2-3*a1*a2*a3*a4+2*a1*a2*a4^2+2*a2^4+3*a2^3*a3-5*a2^3*a4+a2^2*a3^2\
   -5*a2^2*a3*a4+4*a2^2*a4^2-a2*a3^2*a4+2*a2*a3*a4^2-a2*a4^3)
↦     [4]:
↦        _[1]=0
↦     [5]:
↦        _[1]=(a2-a4)
↦     [6]:
↦        [1]:
↦           _[1]=0
↦     [7]:
↦        _[1]=x4
↦        _[2]=x1
↦        _[3]=x3^2
↦        _[4]=x2*x3
↦        _[5]=x2^2
↦     [8]:
↦        1
buildtreetoMaple(T,"Tb","Tb.txt");
```

D.2.9.9 buildtreetoMaple

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: buildtreetoMaple(T, TM, writefile);
T: is the list provided by buildtree,
TM: is the name (string) of the table variable in Maple that will represent the output of buildtree,
writefile: is the name (string) of the file where to write the content.

Return: writes the list provided by buildtree to a file
containing the table representing it in Maple.

Example:

```
LIB "redcgs.lib";
ring R=(0,a1,a2,a3,a4),(x1,x2,x3,x4),dp;
ideal F=x4-a4+a2,
x1+x2+x3+x4-a1-a3-a4,
x1*x3*x4-a1*a3*a4,
x1*x3+x1*x4+x2*x3+x3*x4-a1*a4-a1*a3-a3*a4;
def T=buildtree(F);
finalcases(T);
↦ [1]:
↦    [1]:
↦       0,0
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x4
↦       _[2]=-x1-x2-x3+(a1+a3+a4)
↦       _[3]=x3^2+(-a1-a3-a4)*x3+(a1*a3+a1*a4+a3*a4)
↦    [4]:
↦       _[1]=(a2-a4)
↦       _[2]=(a1*a3*a4)
↦    [5]:
↦       _[1]=0
↦    [6]:
↦       [1]:
↦          _[1]=(a4)
↦          _[2]=(a2)
↦       [2]:
↦          _[1]=(a3)
↦          _[2]=(a2-a4)
↦       [3]:
↦          _[1]=(a2-a4)
↦          _[2]=(a1)
↦    [7]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦    [8]:
↦       1
↦ [2]:
↦    [1]:
↦       0,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=(a2-a4)
↦    [5]:
↦       _[1]=(a1)
↦       _[2]=(a3)
↦       _[3]=(a4)
↦    [6]:
↦       [1]:
↦          _[1]=(a2-a4)
↦    [7]:
↦       _[1]=1
↦    [8]:
```

```
↦       1
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x4+(a2-a4)
↦       _[2]=-x1-x2-x3+(a1+a2+a3)
↦       _[3]=x3^2+(-a2+a4)*x2+(-a1-a2-a3)*x3+(a1*a2+a1*a3+a2^2+a2*a3-a2*a4)
↦       _[4]=(a2-a4)*x2*x3+(a2^2-2*a2*a4+a4^2)*x2+(-a1*a2^2-a1*a2*a3+a1*a2*\
  a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)
↦       _[5]=(a2^2-2*a2*a4+a4^2)*x2^2+(-2*a1*a2^2-a1*a2*a3+3*a1*a2*a4+a1*a3\
  *a4-a1*a4^2-3*a2^3-2*a2^2*a3+7*a2^2*a4+3*a2*a3*a4-5*a2*a4^2-a3*a4^2+a4^3)\
  *x2+(-a1*a2^2-a1*a2*a3+a1*a2*a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)*\
  x3+(a1^2*a2^2+a1^2*a2*a3-a1^2*a2*a4+3*a1*a2^3+4*a1*a2^2*a3-5*a1*a2^2*a4+a\
  1*a2*a3^2-3*a1*a2*a3*a4+2*a1*a2*a4^2+2*a2^4+3*a2^3*a3-5*a2^3*a4+a2^2*a3^2\
  -5*a2^2*a3*a4+4*a2^2*a4^2-a2*a3^2*a4+2*a2*a3*a4^2-a2*a4^3)
↦    [4]:
↦       _[1]=0
↦    [5]:
↦       _[1]=(a2-a4)
↦    [6]:
↦       [1]:
↦          _[1]=0
↦    [7]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦       _[4]=x2*x3
↦       _[5]=x2^2
↦    [8]:
↦       1
buildtreetoMaple(T,"Tb","Tb.txt");
```

D.2.9.10 finalcases

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: finalcases(T);
T is the list provided by buildtree

Return: A list with the CGS determined by buildtree.
Each element of the list represents one segment
of the buildtree CGS.
The list elements have the following structure:
[1]: label (an intvec(1,0,..)) that indicates the position in the buildtree but that is irrelevant for the CGS
[2]: 1 (integer) it is also irrelevant and indicates
that this was a terminal vertex in buildtree.
[3]: the reduced basis of the segment.
[4], [5], [6]: the red-spec of the null and non-null conditions of the segment.
[4] is the null-conditions radical ideal N,
[5] is the non-null polynomials set (ideal) W,

[6] is the set of prime components (ideals) of N.
[7]: is the set of lpp
[8]: poly 1 (irrelevant) is the condition to branch (but no more branch is necessary in the discussion, so 1 is the result.

Note: It can be called having as argument the list output by buildtree

Example:

```
LIB "redcgs.lib";
ring R=(0,a1,a2,a3,a4),(x1,x2,x3,x4),dp;
ideal F=x4-a4+a2, x1+x2+x3+x4-a1-a3-a4, x1*x3*x4-a1*a3*a4, x1*x3+x1*x4+x2*x3+x3*x4-a1
def T=buildtree(F);
finalcases(T);
↦ [1]:
↦    [1]:
↦       0,0
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x4
↦       _[2]=-x1-x2-x3+(a1+a3+a4)
↦       _[3]=x3^2+(-a1-a3-a4)*x3+(a1*a3+a1*a4+a3*a4)
↦    [4]:
↦       _[1]=(a2-a4)
↦       _[2]=(a1*a3*a4)
↦    [5]:
↦       _[1]=0
↦    [6]:
↦       [1]:
↦          _[1]=(a4)
↦          _[2]=(a2)
↦       [2]:
↦          _[1]=(a3)
↦          _[2]=(a2-a4)
↦       [3]:
↦          _[1]=(a2-a4)
↦          _[2]=(a1)
↦    [7]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦    [8]:
↦       1
↦ [2]:
↦    [1]:
↦       0,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=(a2-a4)
↦    [5]:
↦       _[1]=(a1)
↦       _[2]=(a3)
↦       _[3]=(a4)
↦     [6]:
↦       [1]:
↦         _[1]=(a2-a4)
↦     [7]:
↦       _[1]=1
↦     [8]:
↦       1
↦ [3]:
↦     [1]:
↦       1
↦     [2]:
↦       1
↦     [3]:
↦       _[1]=x4+(a2-a4)
↦       _[2]=-x1-x2-x3+(a1+a2+a3)
↦       _[3]=x3^2+(-a2+a4)*x2+(-a1-a2-a3)*x3+(a1*a2+a1*a3+a2^2+a2*a3-a2*a4)
↦       _[4]=(a2-a4)*x2*x3+(a2^2-2*a2*a4+a4^2)*x2+(-a1*a2^2-a1*a2*a3+a1*a2*\
  a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)
↦       _[5]=(a2^2-2*a2*a4+a4^2)*x2^2+(-2*a1*a2^2-a1*a2*a3+3*a1*a2*a4+a1*a3\
  *a4-a1*a4^2-3*a2^3-2*a2^2*a3+7*a2^2*a4+3*a2*a3*a4-5*a2*a4^2-a3*a4^2+a4^3)\
  *x2+(-a1*a2^2-a1*a2*a3+a1*a2*a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)*\
  x3+(a1^2*a2^2+a1^2*a2*a3-a1^2*a2*a4+3*a1*a2^3+4*a1*a2^2*a3-5*a1*a2^2*a4+a\
  1*a2*a3^2-3*a1*a2*a3*a4+2*a1*a2*a4^2+2*a2^4+3*a2^3*a3-5*a2^3*a4+a2^2*a3^2\
  -5*a2^2*a3*a4+4*a2^2*a4^2-a2*a3^2*a4+2*a2*a3*a4^2-a2*a4^3)
↦     [4]:
↦       _[1]=0
↦     [5]:
↦       _[1]=(a2-a4)
↦     [6]:
↦       [1]:
↦         _[1]=0
↦     [7]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦       _[4]=x2*x3
↦       _[5]=x2^2
↦     [8]:
↦       1
```

D.2.9.11 mrcgs

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: mrcgs(F);
F is the ideal from which to obtain the Minimal Reduced CGS. Alternatively, as option: mrcgs(F,L);
where L is a list of the null conditions ideal N, and W the set of non-null polynomials (ideal). If this option is set, the ideals N and W must depend only on the parameters and the parameter space is reduced to V(N) \ V(h), where h=prod(w), for w in W.

A reduced specification of (N,W) will be computed and used to restrict the parameter-space. The output will omit the known restrictions given as option.

Return: The list representing the Minimal Reduced CGS.
The description given here is identical for rcgs and crcgs. The elements of the list T computed by mrcgs are lists representing a rooted tree.
Each element has as the two first entries with the following content:
[1]: The label (intvec) representing the position in the rooted tree: 0 for the root (and this is a special element) i for the root of the segment i
(i,...) for the children of the segment i
[2]: the number of children (int) of the vertex.
There thus three kind of vertices:
1) the root (first element labelled 0),
2) the vertices labelled with a single integer i,
3) the rest of vertices labelled with more indices.
Description of the root. Vertex type 1)
There is a special vertex (the first one) whose content is the following:
[3] lpp of the given ideal
[4] the given ideal
[5] the red-spec of the (optional) given null and non-null conditions (see redspec for the description)
[6] MRCGS (to remember which algorithm has been used). If the algorithm used is rcgs of crcgs then this will be stated at this vertex (RCGS or CRCGS).
Description of vertices type 2). These are the vertices that initiate a segment, and are labelled with a single integer. [3] lpp (ideal) of the reduced basis. If they are repeated lpp's this will correspond to a sheaf.
[4] the reduced basis (ideal) of the segment.
Description of vertices type 3). These vertices have as first label i and descend form vertex i in the position of the label (i,...). They contain moreover a unique prime ideal in the parameters and form ascending chains of ideals.
How is to be read the mrcgs tree? The vertices with an even number of integers in the label are to be considered as additive and those with an odd number of integers in the label are to be considered as subtraction. As an example consider the following vertices: v1=((i),2,lpp,B),
v2=((i,1),2,P_{(i,1)}),
v3=((i,1,1),2,P_{(i,1,1)},
v4=((i,1,1,1),1,P_{(i,1,1,1)},
v5=((i,1,1,1,1),0,P_{(i,1,1,1,1)},
v6=((i,1,1,2),1,P_{(i,1,1,2)},
v7=((i,1,1,2,1),0,P_{(i,1,1,2,1)},
v8=((i,1,2),0,P_{(i,1,2)},
v9=((i,2),1,P_{(i,2)},
v10=((i,2,1),0,P_{(i,2,1)},
They represent the segment:
(V(i,1)\(((V(i,1,1) \ ((V(i,1,1,1) \ V(i,1,1,1,1)) u (V(i,1,1,2) \ V(i,1,1,2,1))))) u V(i,1,2)) u (V(i,2) \ V(i,2,1))
and can also be represented by
(V(i,1) \ (V(i,1,1) u V(i,1,2))) u
(V(i,1,1,1) \ V(i,1,1,1)) u
(V(i,1,1,2) \ V(i,1,1,2,1)) u

(V(i,2) \ V(i,2,1))
where V(i,j,..) = V(P_{(i,j,..)}

Note: There are three fundamental routines in the library: mrcgs, rcgs and crcgs. mrcgs (Minimal Reduced CGS) is an algorithm that packs so much as it is able to do (using algorithms adhoc) the segments with the same lpp, obtaining the minimal number of segments. The hypothesis is that this is also canonical, but for the moment there is no proof of the uniqueness of that minimal packing. Moreover, the segments that are obtained are not locally closed, i.e. there are not always the difference of two varieties, but can be a union of differences.

The output can be visualized using cantreetoMaple, that will write a file with the content of mrcgs that can be read in Maple and plotted using the Maple plotcantree routine of the Monte's dpgb library You can also try the routine cantodiffcgs when the segments are all difference of two varieties to have a simpler view of the output. But it will give an error if the output is not locally closed.

Example:

```
LIB "redcgs.lib";
ring R=(0,b,c,d,e,f),(x,y),dp;
ideal F=x^2+b*y^2+2*c*x*y+2*d*x+2*e*y+f, 2*x+2*c*y+2*d, 2*b*y+2*c*x+2*e;
def T=mrcgs(F);
T;
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       3
↦    [3]:
↦       _[1]=x2
↦       _[2]=x
↦       _[3]=x
↦    [4]:
↦       _[1]=x2+(2c)*xy+(b)*y2+(2d)*x+(2e)*y+(f)
↦       _[2]=2*x+(2c)*y+(2d)
↦       _[3]=(2c)*x+(2b)*y+(2e)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       MRCGS
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
```

```
↦         _[1]=1
↦  [3]:
↦     [1]:
↦        1,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=0
↦  [4]:
↦     [1]:
↦        1,1,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(bd2-bf+c2f-2cde+e2)
↦  [5]:
↦     [1]:
↦        1,1,1,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(cd-e)
↦        _[2]=(b-c2)
↦  [6]:
↦     [1]:
↦        1,1,1,1,1
↦     [2]:
↦        0
↦     [3]:
↦        _[1]=(d2-f)
↦        _[2]=(cf-de)
↦        _[3]=(cd-e)
↦        _[4]=(b-c2)
↦  [7]:
↦     [1]:
↦        2
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=y
↦        _[2]=x
↦     [4]:
↦        _[1]=(b-c2)*y+(-cd+e)
↦        _[2]=(-b+c2)*x+(-bd+ce)
↦  [8]:
↦     [1]:
↦        2,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(bd2-bf+c2f-2cde+e2)
↦  [9]:
↦     [1]:
```

```
↦       2,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(cd-e)
↦       _[2]=(b-c2)
↦ [10]:
↦    [1]:
↦       3
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x
↦    [4]:
↦       _[1]=x+(c)*y+(d)
↦ [11]:
↦    [1]:
↦       3,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(d2-f)
↦       _[2]=(cf-de)
↦       _[3]=(cd-e)
↦       _[4]=(b-c2)
↦ [12]:
↦    [1]:
↦       3,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
cantreetoMaple(T,"Tm","Tm.txt");
//cantodiffcgs(T); // has non locally closed segments
ring R=(0,a1,a2,a3,a4),(x1,x2,x3,x4),dp;
↦ // ** redefining R (ring R=(0,a1,a2,a3,a4),(x1,x2,x3,x4),dp;) ./examples/\
   mrcgs.sing:8
ideal F2=x4-a4+a2, x1+x2+x3+x4-a1-a3-a4, x1*x3*x4-a1*a3*a4, x1*x3+x1*x4+x2*x3+x3*x4-a
def T2=mrcgs(F2);
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);      // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);     // global ring K[x,a] with\
    product order)
T2;
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       3
↦    [3]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x1*x3*x4
↦          _[4]=x1*x3
↦       [4]:
↦          _[1]=x4+(a2-a4)
↦          _[2]=x1+x2+x3+x4+(-a1-a3-a4)
↦          _[3]=x1*x3*x4+(-a1*a3*a4)
↦          _[4]=x1*x3+x2*x3+x1*x4+x3*x4+(-a1*a3-a1*a4-a3*a4)
↦       [5]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             _[1]=0
↦          [3]:
↦             [1]:
↦                _[1]=0
↦       [6]:
↦          MRCGS
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦       [3]:
↦          _[1]=x4
↦          _[2]=x1
↦          _[3]=x3^2
↦          _[4]=x2*x3
↦          _[5]=x2^2
↦       [4]:
↦          _[1]=x4+(a2-a4)
↦          _[2]=-x1-x2-x3+(a1+a2+a3)
↦          _[3]=x3^2+(-a2+a4)*x2+(-a1-a2-a3)*x3+(a1*a2+a1*a3+a2^2+a2*a3-a2*a4)
↦          _[4]=(a2-a4)*x2*x3+(a2^2-2*a2*a4+a4^2)*x2+(-a1*a2^2-a1*a2*a3+a1*a2*\
   a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)
↦          _[5]=(a2^2-2*a2*a4+a4^2)*x2^2+(-2*a1*a2^2-a1*a2*a3+3*a1*a2*a4+a1*a3\
   *a4-a1*a4^2-3*a2^3-2*a2^2*a3+7*a2^2*a4+3*a2*a3*a4-5*a2*a4^2-a3*a4^2+a4^3)\
   *x2+(-a1*a2^2-a1*a2*a3+a1*a2*a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)*\
   x3+(a1^2*a2^2+a1^2*a2*a3-a1^2*a2*a4+3*a1*a2^3+4*a1*a2^2*a3-5*a1*a2^2*a4+a\
   1*a2*a3^2-3*a1*a2*a3*a4+2*a1*a2*a4^2+2*a2^4+3*a2^3*a3-5*a2^3*a4+a2^2*a3^2\
   -5*a2^2*a3*a4+4*a2^2*a4^2-a2*a3^2*a4+2*a2*a3*a4^2-a2*a4^3)
↦    [3]:
↦       [1]:
↦          1,1
↦       [2]:
↦          1
↦       [3]:
↦          _[1]=0
↦    [4]:
↦       [1]:
↦          1,1,1
↦       [2]:
↦          0
↦       [3]:
↦          _[1]=(a2-a4)
↦ [5]:
↦    [1]:
↦       2
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=1
↦ [6]:
↦    [1]:
↦       2,1
↦    [2]:
↦       3
↦    [3]:
↦       _[1]=(a2-a4)
↦ [7]:
↦    [1]:
↦       2,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(a4)
↦       _[2]=(a2)
↦ [8]:
↦    [1]:
↦       2,1,2
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(a3)
↦       _[2]=(a2-a4)
↦ [9]:
↦    [1]:
↦       2,1,3
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(a2-a4)
↦       _[2]=(a1)
↦ [10]:
↦    [1]:
↦       3
↦    [2]:
↦       3
↦    [3]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦    [4]:
↦       _[1]=x4
↦       _[2]=-x1-x2-x3+(a1+a3+a4)
↦       _[3]=x3^2+(-a1-a3-a4)*x3+(a1*a3+a1*a4+a3*a4)
↦ [11]:
↦    [1]:
↦       3,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(a4)
↦       _[2]=(a2)
↦ [12]:
↦    [1]:
↦       3,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
↦ [13]:
↦    [1]:
↦       3,2
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(a3)
↦       _[2]=(a2-a4)
↦ [14]:
↦    [1]:
↦       3,2,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
↦ [15]:
↦    [1]:
↦       3,3
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(a2-a4)
↦       _[2]=(a1)
↦ [16]:
↦    [1]:
↦       3,3,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
cantreetoMaple(T2,"T2m","T2m.txt");
cantodiffcgs(T2);
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
```

```
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       3
↦    [3]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x1*x3*x4
↦       _[4]=x1*x3
↦    [4]:
↦       _[1]=x4+(a2-a4)
↦       _[2]=x1+x2+x3+x4+(-a1-a3-a4)
↦       _[3]=x1*x3*x4+(-a1*a3*a4)
↦       _[4]=x1*x3+x2*x3+x1*x4+x3*x4+(-a1*a3-a1*a4-a3*a4)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       MRCGS
↦ [2]:
↦    [1]:
↦       _[1]=x4
↦       _[2]=x1
↦       _[3]=x3^2
↦       _[4]=x2*x3
↦       _[5]=x2^2
↦    [2]:
↦       _[1]=x4+(a2-a4)
↦       _[2]=-x1-x2-x3+(a1+a2+a3)
↦       _[3]=x3^2+(-a2+a4)*x2+(-a1-a2-a3)*x3+(a1*a2+a1*a3+a2^2+a2*a3-a2*a4)
↦       _[4]=(a2-a4)*x2*x3+(a2^2-2*a2*a4+a4^2)*x2+(-a1*a2^2-a1*a2*a3+a1*a2*\
  a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)
↦       _[5]=(a2^2-2*a2*a4+a4^2)*x2^2+(-2*a1*a2^2-a1*a2*a3+3*a1*a2*a4+a1*a3\
  *a4-a1*a4^2-3*a2^3-2*a2^2*a3+7*a2^2*a4+3*a2*a3*a4-5*a2*a4^2-a3*a4^2+a4^3)\
  *x2+(-a1*a2^2-a1*a2*a3+a1*a2*a4-a2^3-a2^2*a3+2*a2^2*a4+a2*a3*a4-a2*a4^2)*\
  x3+(a1^2*a2^2+a1^2*a2*a3-a1^2*a2*a4+3*a1*a2^3+4*a1*a2^2*a3-5*a1*a2^2*a4+a\
  1*a2*a3^2-3*a1*a2*a3*a4+2*a1*a2*a4^2+2*a2^4+3*a2^3*a3-5*a2^3*a4+a2^2*a3^2\
  -5*a2^2*a3*a4+4*a2^2*a4^2-a2*a3^2*a4+2*a2*a3*a4^2-a2*a4^3)
↦    [3]:
↦       _[1]=0
↦    [4]:
```

```
↦        _[1]=(a2-a4)
↦  [3]:
↦     [1]:
↦        _[1]=1
↦     [2]:
↦        _[1]=1
↦     [3]:
↦        _[1]=(a2-a4)
↦     [4]:
↦        _[1]=(a2-a4)
↦        _[2]=(a1*a3*a4)
↦  [4]:
↦     [1]:
↦        _[1]=x4
↦        _[2]=x1
↦        _[3]=x3^2
↦     [2]:
↦        _[1]=x4
↦        _[2]=-x1-x2-x3+(a1+a3+a4)
↦        _[3]=x3^2+(-a1-a3-a4)*x3+(a1*a3+a1*a4+a3*a4)
↦     [3]:
↦        _[1]=(a2-a4)
↦        _[2]=(a1*a3*a4)
↦     [4]:
↦        _[1]=1
```

D.2.9.12 rcgs

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: rcgs(F);
F is the ideal from which to obtain the Reduced CGS.
rcgs(F,L);
where L is a list of the null conditions ideal N, and W the set of non-null polynomials (ideal). If this option is set, the ideals N and W must depend only on the parameters and the parameter space is reduced to V(N) \ V(h), where h=prod(w), for w in W.
A reduced specification of (N,W) will be computed and used to restrict the parameter-space. The output will omit the known restrictions given as option.

Return: The list representing the Reduced CGS.
The description given here is analogous as for mrcgs and crcgs. The elements of the list T computed by rcgs are lists representing a rooted tree.
Each element has as the two first entries with the following content:
[1]: The label (intvec) representing the position in the rooted tree: 0 for the root (and this is a special element) i for the root of the segment i
(i,...) for the children of the segment i
[2]: the number of children (int) of the vertex.
There thus three kind of vertices:
1) the root (first element labelled 0),
2) the vertices labelled with a single integer i,
3) the rest of vertices labelled with more indices.
Description of the root. Vertex type 1)
There is a special vertex (the first one) whose content is the following:

[3] lpp of the given ideal
[4] the given ideal
[5] the red-spec of the (optional) given null and non-null conditions (see redspec for the description)
[6] RCGS (to remember which algorithm has been used). If the algorithm used is mrcgs or crcgs then this will be stated at this vertex (mrcgs or CRCGS).
Description of vertices type 2). These are the vertices that initiate a segment, and are labelled with a single integer. [3] lpp (ideal) of the reduced basis. If they are repeated lpp's this will correspond to a sheaf.
[4] the reduced basis (ideal) of the segment.
Description of vertices type 3). These vertices have as first label i and descend form vertex i in the position of the label (i,...). They contain moreover a unique prime ideal in the parameters and form ascending chains of ideals.
How is to be read the rcgs tree? The vertices with an even number of integers in the label are to be considered as additive and those with an odd number of integers in the label are to be considered as subtraction. As an example consider the following vertices: v1=((i),2,lpp,B),
v2=((i,1),2,P_{(i,1)}),
v3=((i,1,1),0,P_{(i,1,1)}, v4=((i,1,2),0,P_{(i,1,1)}), v5=((i,2),2,P_{(i,2)},
v6=((i,2,1),0,P_{(i,2,1)}, v7=((i,2,2),0,P_{(i,2,2)}
They represent the segment:
(V(i,1)\(V(i,1,1) u V(i,1,2))) u
(V(i,2)\(V(i,2,1) u V(i,2,2)))
where V(i,j,..) = V(P_{(i,j,..)}

Note: There are three fundamental routines in the library: mrcgs, rcgs and crcgs. rcgs (Reduced CGS) is an algorithm that first homogenizes the basis of the given ideal then applies mrcgs and finally de-homogenizes and reduces the resulting bases. (See the note of mrcgs). As a result of Wibmer's Theorem, the resulting segments are locally closed (i.e. difference of varieties). Nevertheless, the output is not completely canonical as the homogeneous ideal considered is not the homogenized ideal of the given ideal but only the ideal obtained by homogenizing the given basis.

The output can be visualized using cantreetoMaple, that will write a file with the content of mrcgs that can be read in Maple and plotted using the Maple plotcantree routine of the Monte's dpgb library You can also use the routine cantodiffcgs as the segments are all difference of two varieties to have a simpler view of the output.

Example:

```
LIB "redcgs.lib";
ring R=(0,b,c,d,e,f),(x,y),dp;
ideal F=x^2+b*y^2+2*c*x*y+2*d*x+2*e*y+f, 2*x+2*c*y+2*d, 2*b*y+2*c*x+2*e;
def T=rcgs(F);
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);      // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);     // global ring K[x,a] with\
   product order)
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);      // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);     // global ring K[x,a] with\
   product order)
T;
↦ [1]:
```

```
↦      [1]:
↦         0
↦      [2]:
↦         5
↦      [3]:
↦         _[1]=x2
↦         _[2]=x
↦         _[3]=x
↦      [4]:
↦         _[1]=x2+(2c)*xy+(b)*y2+(2d)*x+(2e)*y+(f)
↦         _[2]=2*x+(2c)*y+(2d)
↦         _[3]=(2c)*x+(2b)*y+(2e)
↦      [5]:
↦         [1]:
↦            _[1]=0
↦         [2]:
↦            _[1]=0
↦         [3]:
↦            [1]:
↦               _[1]=0
↦      [6]:
↦         RCGS
↦   [2]:
↦      [1]:
↦         1
↦      [2]:
↦         1
↦      [3]:
↦         _[1]=1
↦      [4]:
↦         _[1]=1
↦   [3]:
↦      [1]:
↦         1,1
↦      [2]:
↦         2
↦      [3]:
↦         _[1]=0
↦   [4]:
↦      [1]:
↦         1,1,1
↦      [2]:
↦         0
↦      [3]:
↦         _[1]=(bd2-bf+c2f-2cde+e2)
↦   [5]:
↦      [1]:
↦         1,1,2
↦      [2]:
↦         0
↦      [3]:
↦         _[1]=(b-c2)
↦   [6]:
```

```
↦     [1]:
↦        2
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=y
↦        _[2]=x
↦     [4]:
↦        _[1]=(b-c2)*y+(-cd+e)
↦        _[2]=(-b+c2)*x+(-bd+ce)
↦  [7]:
↦     [1]:
↦        2,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(bd2-bf+c2f-2cde+e2)
↦  [8]:
↦     [1]:
↦        2,1,1
↦     [2]:
↦        0
↦     [3]:
↦        _[1]=(cd-e)
↦        _[2]=(b-c2)
↦  [9]:
↦     [1]:
↦        3
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=1
↦     [4]:
↦        _[1]=1
↦  [10]:
↦     [1]:
↦        3,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(b-c2)
↦  [11]:
↦     [1]:
↦        3,1,1
↦     [2]:
↦        0
↦     [3]:
↦        _[1]=(cd-e)
↦        _[2]=(b-c2)
↦  [12]:
↦     [1]:
↦        4
↦     [2]:
```

```
↦         1
↦      [3]:
↦         _[1]=1
↦      [4]:
↦         _[1]=1
↦   [13]:
↦      [1]:
↦         4,1
↦      [2]:
↦         1
↦      [3]:
↦         _[1]=(cd-e)
↦         _[2]=(b-c2)
↦   [14]:
↦      [1]:
↦         4,1,1
↦      [2]:
↦         0
↦      [3]:
↦         _[1]=(d2-f)
↦         _[2]=(cf-de)
↦         _[3]=(cd-e)
↦         _[4]=(b-c2)
↦   [15]:
↦      [1]:
↦         5
↦      [2]:
↦         1
↦      [3]:
↦         _[1]=x
↦      [4]:
↦         _[1]=x+(c)*y+(d)
↦   [16]:
↦      [1]:
↦         5,1
↦      [2]:
↦         1
↦      [3]:
↦         _[1]=(d2-f)
↦         _[2]=(cf-de)
↦         _[3]=(cd-e)
↦         _[4]=(b-c2)
↦   [17]:
↦      [1]:
↦         5,1,1
↦      [2]:
↦         0
↦      [3]:
↦         _[1]=1
cantreetoMaple(T,"Tr","Tr.txt");
cantodiffcgs(T);
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
```

```
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       5
↦    [3]:
↦       _[1]=x^2
↦       _[2]=x
↦       _[3]=x
↦    [4]:
↦       _[1]=x^2+(2*c)*x*y+(b)*y^2+(2*d)*x+(2*e)*y+(f)
↦       _[2]=2*x+(2*c)*y+(2*d)
↦       _[3]=(2*c)*x+(2*b)*y+(2*e)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       RCGS
↦ [2]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       _[1]=0
↦    [4]:
↦       _[1]=(b^2*d^2-b^2*f-b*c^2*d^2+2*b*c^2*f-2*b*c*d*e+b*e^2-c^4*f+2*c^3\
  *d*e-c^2*e^2)
↦ [3]:
↦    [1]:
```

```
↦          _[1]=y
↦          _[2]=x
↦       [2]:
↦          _[1]=(b-c^2)*y+(-c*d+e)
↦          _[2]=(-b+c^2)*x+(-b*d+c*e)
↦       [3]:
↦          _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦       [4]:
↦          _[1]=(c*d-e)
↦          _[2]=(b-c^2)
↦    [4]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=1
↦       [3]:
↦          _[1]=(b-c^2)
↦       [4]:
↦          _[1]=(c*d-e)
↦          _[2]=(b-c^2)
↦    [5]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=1
↦       [3]:
↦          _[1]=(c*d-e)
↦          _[2]=(b-c^2)
↦       [4]:
↦          _[1]=(d^2-f)
↦          _[2]=(c*f-d*e)
↦          _[3]=(c*d-e)
↦          _[4]=(b-c^2)
↦    [6]:
↦       [1]:
↦          _[1]=x
↦       [2]:
↦          _[1]=x+(c)*y+(d)
↦       [3]:
↦          _[1]=(d^2-f)
↦          _[2]=(c*f-d*e)
↦          _[3]=(c*d-e)
↦          _[4]=(b-c^2)
↦       [4]:
↦          _[1]=1
```

D.2.9.13 crcgs

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: crcgs(F);
F is the ideal from which to obtain the Canonical Reduced CGS. crcgs(F,L);
where L is a list of the null conditions ideal N, and W the set of non-null polynomials (ideal). If this option is set, the ideals N and W must depend only on the parameters

and the parameter space is reduced to V(N) \ V(h), where h=prod(w), for w in W.

A reduced specification of (N,W) will be computed and used to restrict the parameter-space. The output will omit the known restrictions given as option.

Return: The list representing the Canonical Reduced CGS.

The description given here is identical for mrcgs and rcgs. The elements of the list T computed by crcgs are lists representing a rooted tree.

Each element has as the two first entries with the following content:

[1]: The label (intvec) representing the position in the rooted tree: 0 for the root (and this is a special element) i for the root of the segment i

(i,...) for the children of the segment i

[2]: the number of children (int) of the vertex.

There thus three kind of vertices:

1) the root (first element labelled 0),

2) the vertices labelled with a single integer i,

3) the rest of vertices labelled with more indices.

Description of the root. Vertex type 1)

There is a special vertex (the first one) whose content is the following:

[3] lpp of the given ideal

[4] the given ideal

[5] the red-spec of the (optional) given null and non-null conditions (see redspec for the description)

[6] mrcgs (to remember which algorithm has been used). If the algorithm used is rcgs of crcgs then this will be stated at this vertex (RCGS or CRCGS).

Description of vertices type 2). These are the vertices that initiate a segment, and are labelled with a single integer. [3] lpp (ideal) of the reduced basis. If they are repeated lpp's this will correspond to a sheaf.

[4] the reduced basis (ideal) of the segment.

Description of vertices type 3). These vertices have as first label i and descend form vertex i in the position of the label (i,...). They contain moreover a unique prime ideal in the parameters and form ascending chains of ideals.

How is to be read the mrcgs tree? The vertices with an even number of integers in the label are to be considered as additive and those with an odd number of integers in the label are to be considered as subtraction. As an example consider the following vertices: v1=((i),2,lpp,B),

v2=((i,1),2,P_{(i,1)}),

v3=((i,1,1),0,P_{(i,1,1)}, v4=((i,1,2),0,P_{(i,1,1)}), v5=((i,2),2,P_{(i,2)},

v6=((i,2,1),0,P_{(i,2,1)}, v7=((i,2,2),0,P_{(i,2,2)}

They represent the segment:

(V(i,1)\(V(i,1,1) u V(i,1,2))) u

(V(i,2)\(V(i,2,1) u V(i,2,2)))

where V(i,j,..) = V(P_{(i,j,..)}

Note: There are three fundamental routines in the library: mrcgs, rcgs and crcgs. crcgs (Canonical Reduced CGS) is an algorithm that first homogenizes the the given ideal then applies mrcgs and finally de-homogenizes and reduces the resulting bases. (See the note of mrcgs). As a result of Wibmer's Theorem, the resulting segments are locally closed (i.e. difference of varieties) and the partition is canonical as the homogenized ideal is uniquely associated to the given ideal not depending of the given basis.

Nevertheless the computations to do are usually more time consuming and so it is preferable to compute first the rcgs and only if it success you can try crcgs.

The output can be visualized using cantreetoMaple, that will write a file with the content of crcgs that can be read in Maple and plotted using the Maple plotcantree routine of the Monte's dpgb library You can also use the routine cantodiffcgs as the segments are all difference of two varieties to have a simpler view of the output.

Example:

```
LIB "redcgs.lib";
ring R=(0,b,c,d,e,f),(x,y),dp;
ideal F=x^2+b*y^2+2*c*x*y+2*d*x+2*e*y+f, 2*x+2*c*y+2*d, 2*b*y+2*c*x+2*e;
def T=crcgs(F);
↦ // ** redefining @R (  exportto(Top,@R);       // global ring K[a][x])
↦ // ** redefining @R (  exportto(Top,@R);       // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);       // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);      // global ring K[x,a] with\
    product order)
↦ // ** NP2 is no standard basis
↦ // ** NP2 is no standard basis
↦ // ** redefining @R (  exportto(Top,@R);       // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);       // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);      // global ring K[x,a] with\
    product order)
T;
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       4
↦    [3]:
↦       _[1]=1
↦       _[2]=y
↦       _[3]=y
↦       _[4]=x
↦    [4]:
↦       _[1]=(bd2-bf+c2f-2cde+e2)
↦       _[2]=(cd-e)*y+(d2-f)
↦       _[3]=(b-c2)*y+(-cd+e)
↦       _[4]=x+(c)*y+(d)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       CRCGS
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
```

```
↦        _[1]=1
↦     [4]:
↦        _[1]=1
↦  [3]:
↦     [1]:
↦        1,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=0
↦  [4]:
↦     [1]:
↦        1,1,1
↦     [2]:
↦        0
↦     [3]:
↦        _[1]=(bd2-bf+c2f-2cde+e2)
↦  [5]:
↦     [1]:
↦        2
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=y
↦        _[2]=x
↦     [4]:
↦        _[1]=(b-c2)*y+(-cd+e)
↦        _[2]=(b-c2)*x+(bd-ce)
↦  [6]:
↦     [1]:
↦        2,1
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=(bd2-bf+c2f-2cde+e2)
↦  [7]:
↦     [1]:
↦        2,1,1
↦     [2]:
↦        0
↦     [3]:
↦        _[1]=(cd-e)
↦        _[2]=(b-c2)
↦  [8]:
↦     [1]:
↦        3
↦     [2]:
↦        1
↦     [3]:
↦        _[1]=1
↦     [4]:
↦        _[1]=1
↦  [9]:
```

```
↦    [1]:
↦       3,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(cd-e)
↦       _[2]=(b-c2)
↦ [10]:
↦    [1]:
↦       3,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(d2-f)
↦       _[2]=(cf-de)
↦       _[3]=(cd-e)
↦       _[4]=(b-c2)
↦ [11]:
↦    [1]:
↦       4
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x
↦    [4]:
↦       _[1]=x+(c)*y+(d)
↦ [12]:
↦    [1]:
↦       4,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(d2-f)
↦       _[2]=(cf-de)
↦       _[3]=(cd-e)
↦       _[4]=(b-c2)
↦ [13]:
↦    [1]:
↦       4,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
cantreetoMaple(T,"Tc","Tc.txt");
cantodiffcgs(T);
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
```

```
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
  0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
  1
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       4
↦    [3]:
↦       _[1]=1
↦       _[2]=y
↦       _[3]=y
↦       _[4]=x
↦    [4]:
↦       _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦       _[2]=(c*d-e)*y+(d^2-f)
↦       _[3]=(b-c^2)*y+(-c*d+e)
↦       _[4]=x+(c)*y+(d)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       CRCGS
↦ [2]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       _[1]=0
↦    [4]:
↦       _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦ [3]:
↦    [1]:
↦       _[1]=y
↦       _[2]=x
↦    [2]:
↦       _[1]=(b-c^2)*y+(-c*d+e)
↦       _[2]=(b-c^2)*x+(b*d-c*e)
↦    [3]:
↦       _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦    [4]:
↦       _[1]=(c*d-e)
↦        _[2]=(b-c^2)
↦ [4]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦    [3]:
↦       _[1]=(c*d-e)
↦       _[2]=(b-c^2)
↦    [4]:
↦       _[1]=(d^2-f)
↦       _[2]=(c*f-d*e)
↦       _[3]=(c*d-e)
↦       _[4]=(b-c^2)
↦ [5]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=x+(c)*y+(d)
↦    [3]:
↦       _[1]=(d^2-f)
↦       _[2]=(c*f-d*e)
↦       _[3]=(c*d-e)
↦       _[4]=(b-c^2)
↦    [4]:
↦       _[1]=1
```

D.2.9.14 cantodiffcgs

Procedure from library `redcgs.lib` (see Section D.2.9 [redcgs_lib], page 897).

Usage: canttodiffcgs(T);
T: is the list provided by mrcgs or crcgs or crcgs,

Return: The list transforming the content of these routines to a simpler output where each segment corresponds to a single element of the list that is described as difference of two varieties.

The first element of the list is identical to the first element of the list provided by the corresponding cgs algorithm, and contains general information on the call (see mrcgs).
The remaining elements are lists of 4 elements,
representing segments. These elements are
[1]: the lpp of the segment
[2]: the basis of the segment
[3]; the ideal of the first variety (radical)
[4]; the ideal of the second variety (radical)
The segment is V([3]) \ V([4]).

Note: It can be called from the output of mrcgs or rcgs of crcgs

Example:

```
LIB "redcgs.lib";
ring R=(0,b,c,d,e,f),(x,y),dp;
ideal F=x^2+b*y^2+2*c*x*y+2*d*x+2*e*y+f, 2*x+2*c*y+2*d, 2*b*y+2*c*x+2*e;
def T=crcgs(F);
```

```
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);      // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);     // global ring K[x,a] with\
   product order)
↦ // ** NP2 is no standard basis
↦ // ** NP2 is no standard basis
↦ // ** redefining @R (  exportto(Top,@R);      // global ring K[a][x])
↦ // ** redefining @P (  exportto(Top,@P);      // global ring K[a])
↦ // ** redefining @RP (  exportto(Top,@RP);     // global ring K[x,a] with\
   product order)
T;
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       4
↦    [3]:
↦       _[1]=1
↦       _[2]=y
↦       _[3]=y
↦       _[4]=x
↦    [4]:
↦       _[1]=(bd2-bf+c2f-2cde+e2)
↦       _[2]=(cd-e)*y+(d2-f)
↦       _[3]=(b-c2)*y+(-cd+e)
↦       _[4]=x+(c)*y+(d)
↦    [5]:
↦       [1]:
↦          _[1]=0
↦       [2]:
↦          _[1]=0
↦       [3]:
↦          [1]:
↦             _[1]=0
↦    [6]:
↦       CRCGS
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=1
↦ [3]:
↦    [1]:
↦       1,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=0
↦ [4]:
↦    [1]:
↦       1,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(bd2-bf+c2f-2cde+e2)
↦ [5]:
↦    [1]:
↦       2
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=y
↦       _[2]=x
↦    [4]:
↦       _[1]=(b-c2)*y+(-cd+e)
↦       _[2]=(b-c2)*x+(bd-ce)
↦ [6]:
↦    [1]:
↦       2,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(bd2-bf+c2f-2cde+e2)
↦ [7]:
↦    [1]:
↦       2,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(cd-e)
↦       _[2]=(b-c2)
↦ [8]:
↦    [1]:
↦       3
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=1
↦    [4]:
↦       _[1]=1
↦ [9]:
↦    [1]:
↦       3,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(cd-e)
↦       _[2]=(b-c2)
↦ [10]:
↦    [1]:
↦       3,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=(d2-f)
↦       _[2]=(cf-de)
↦       _[3]=(cd-e)
↦       _[4]=(b-c2)
↦ [11]:
↦    [1]:
↦       4
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=x
↦    [4]:
↦       _[1]=x+(c)*y+(d)
↦ [12]:
↦    [1]:
↦       4,1
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=(d2-f)
↦       _[2]=(cf-de)
↦       _[3]=(cd-e)
↦       _[4]=(b-c2)
↦ [13]:
↦    [1]:
↦       4,1,1
↦    [2]:
↦       0
↦    [3]:
↦       _[1]=1
cantreetoMaple(T,"Tc","Tc.txt");
cantodiffcgs(T);
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ // ** redefining NP (    def NP=imap(RR,N);) redcgs.lib::cantodiffcgs:396\
   0
↦ // ** redefining MP (    def MP=imap(RR,M);) redcgs.lib::cantodiffcgs:396\
   1
↦ [1]:
↦    [1]:
```

```
↦          0
↦       [2]:
↦          4
↦       [3]:
↦          _[1]=1
↦          _[2]=y
↦          _[3]=y
↦          _[4]=x
↦       [4]:
↦          _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦          _[2]=(c*d-e)*y+(d^2-f)
↦          _[3]=(b-c^2)*y+(-c*d+e)
↦          _[4]=x+(c)*y+(d)
↦       [5]:
↦          [1]:
↦             _[1]=0
↦          [2]:
↦             _[1]=0
↦          [3]:
↦             [1]:
↦                _[1]=0
↦       [6]:
↦          CRCGS
↦    [2]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=1
↦       [3]:
↦          _[1]=0
↦       [4]:
↦          _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦    [3]:
↦       [1]:
↦          _[1]=y
↦          _[2]=x
↦       [2]:
↦          _[1]=(b-c^2)*y+(-c*d+e)
↦          _[2]=(b-c^2)*x+(b*d-c*e)
↦       [3]:
↦          _[1]=(b*d^2-b*f+c^2*f-2*c*d*e+e^2)
↦       [4]:
↦          _[1]=(c*d-e)
↦          _[2]=(b-c^2)
↦    [4]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=1
↦       [3]:
↦          _[1]=(c*d-e)
↦          _[2]=(b-c^2)
↦       [4]:
```

```
↦        _[1]=(d^2-f)
↦        _[2]=(c*f-d*e)
↦        _[3]=(c*d-e)
↦        _[4]=(b-c^2)
↦ [5]:
↦    [1]:
↦        _[1]=x
↦    [2]:
↦        _[1]=x+(c)*y+(d)
↦    [3]:
↦        _[1]=(d^2-f)
↦        _[2]=(c*f-d*e)
↦        _[3]=(c*d-e)
↦        _[4]=(b-c^2)
↦    [4]:
↦        _[1]=1
```

D.2.10 random_lib

Library: random.lib

Purpose: Creating Random and Sparse Matrices, Ideals, Polys

Procedures:

D.2.10.1 genericid

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: genericid(id[,p,b]); id ideal/module, p,b integers

Return: system of generators of id which are generic, sparse, triagonal linear combinations of given generators with coefficients in [1,b] and sparsety p percent, bigger p being sparser (default: p=75, b=30000)

Note: For performance reasons try small bound b in characteristic 0

Example:

```
LIB "random.lib";
ring r=0,(t,x,y,z),ds;
ideal i= x3+y4,z4+yx,t+x+y+z;
genericid(i,0,10);
↦ _[1]=3t+3x+3y+3z+2xy+x3+y4+2z4
↦ _[2]=4t+4x+4y+4z+xy+z4
↦ _[3]=t+x+y+z
module m=[x,0,0,0],[0,y2,0,0],[0,0,z3,0],[0,0,0,t4];
print(genericid(m));
↦ x,      0,      0, 0,
↦ 17904y2,y2,     0, 0,
↦ 0,      24170z3,z3,0,
↦ 0,      0,      0, t4
```

D.2.10.2 randomid

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: randomid(id[,k,b]); id ideal/module, b,k integers

Return: ideal/module having k generators which are random linear combinations of generators of id with coefficients in the interval [-b,b] (default: b=30000, k=size(id))

Note: For performance reasons try small bound b in characteristic 0

Example:

```
LIB "random.lib";
ring r=0,(x,y,z),dp;
randomid(maxideal(2),2,9);
↦ _[1]=-5x2-9xy+6y2-8xz-8yz+4z2
↦ _[2]=-9xy+2y2+xz+yz-z2
module m=[x,0,1],[0,y2,0],[y,0,z3];
show(randomid(m));
↦ // module, 3 generator(s)
↦ [1369x-11685y,-4481y2,-11685z3+1369]
↦ [-642x-13756y,25342y2,-13756z3-642]
↦ [2536x-6355y,8285y2,-6355z3+2536]
```

D.2.10.3 randommat

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: randommat(n,m[,id,b]); n,m,b integers, id ideal

Return: nxm matrix, entries are random linear combinations of elements of id and coefficients in [-b,b]
[default: (id,b) = (maxideal(1),30000)]

Note: For performance reasons try small bound b in char 0

Example:

```
LIB "random.lib";
ring r=0,(x,y,z),dp;
matrix A=randommat(3,3,maxideal(2),9);
print(A);
↦ 9x2-2xy-8y2-9xz+yz+4z2, 9x2-4xy+y2-5xz+6yz-z2,   8x2+xy-9y2+2yz-8z2,
↦ -x2+5xy-8y2-7xz+4yz-3z2,x2+xy-4y2-xz+5z2,        5x2-8xy+8y2+6xz+yz+7z2,
↦ 4x2-5xy-6y2-4yz-5z2,    -4x2-6xy-4y2-8xz+3yz+5z2,2x2+3xy+y2+4xz-3yz+2z2
A=randommat(2,3);
print(A);
↦ 15276x+9897y+7526z,  6495x-24178y+11295z,-5745x-14754y+15979z,
↦ 20788x-28366y-20283z,24911x-10978y+3341z,12412x+11216y+15344z
```

D.2.10.4 sparseid

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparseid(k,u[,o,p,b]); k,u,o,p,b integers

Return: ideal having k generators, each of degree d, u<=d<=o, p percent of terms in degree d are 0, the remaining have random coefficients in the interval [1,b], (default: o=u, p=75, b=30000)

Example:

```
LIB "random.lib";
ring r = 0,(a,b,c,d),ds;
sparseid(2,3);"";
↦ _[1]=12773a3+24263a2c+20030abc+17904b2c+26359c3
↦ _[2]=24004a3+6204b2c+24170bc2+19505c2d+21962bd2
↦
sparseid(3,0,4,90,9);
↦ _[1]=1+4a2+8b2c+3c3+4a3b+4a2b2+5abc2+3ac3
↦ _[2]=a+a2+7ab2+6a2c+3c3+5a3b+9ab3+2c4+3c3d+8ad3
↦ _[3]=5a+ab+2ac2+2b3c+8abcd
```

D.2.10.5 sparsematrix

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparsematrix(n,m,o[,u,pe,pp,b]); n,m,o,u,pe,pp,b integers

Return: nxm matrix, about pe percent of the entries are 0, the remaining are random polynomials of degree d, u<=d<=o, with pp percent of the terms being 0, the remaining have random coefficients in the interval [1,b] [default: (pe,u,pp,b) = (0,50,75,100)]

Example:

```
LIB "random.lib";
ring r = 0,(a,b,c,d),dp;
// sparse matrix of sparse polys of degree <=2:
print(sparsematrix(3,4,2));"";
↦ 17a2+30ab+94bc+19b+45d,88a2+44bc+13d2+31a,0,                           0,
↦ 0,                     0,                 6c2+16b+64c+76,              0,
↦ 14ab+20bc+79cd+30b,    32a2+97bc+5b,      23bc+73c2+ad+48cd+73b+59d+25,0
↦
// dense matrix of sparse linear forms:
print(sparsematrix(3,3,1,1,0,55,9));
↦ 9a+5b+9c,2a+9d,2d,
↦ 7c+d,    a+6b, 2b+2d,
↦ 9b+7c+8d,9b+9d,5a
```

D.2.10.6 sparsemat

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparsemat(n,m[,p,b]); n,m,p,b integers

Return: nxm integer matrix, p percent of the entries are 0, the remaining are random coefficients >=1 and <= b; [defaults: (p,b) = (75,1)]

Example:

```
LIB "random.lib";
sparsemat(5,5);"";
↦ 0,0,0,0,0,
↦ 0,1,0,0,1,
↦ 0,0,0,1,0,
↦ 0,1,0,0,0,
↦ 0,1,0,1,1
↦
sparsemat(5,5,95);"";
```

```
↦ 1,0,0,0,0,
↦ 0,0,0,0,0,
↦ 0,0,0,0,0,
↦ 0,0,0,0,0,
↦ 0,0,0,1,0
↦
sparsemat(5,5,5);"";
↦ 1,1,1,1,1,
↦ 1,1,1,1,1,
↦ 1,1,1,1,1,
↦ 1,0,1,1,1,
↦ 1,1,1,1,0
↦
sparsemat(5,5,50,100);
↦ 0,17,24,80,0,
↦ 0,13,30,45,0,
↦ 19,0,0,0,0,
↦ 93,0,23,0,69,
↦ 0,88,44,31,0
```

D.2.10.7 sparsepoly

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparsepoly(u[,o,p,b]); u,o,p,b integers

Return: poly having only terms in degree d, u<=d<=o, p percentage of the terms in degree d are 0, the remaining have random coefficients in [1,b), (defaults: o=u, p=75, b=30000)

Example:

```
LIB "random.lib";
ring r=0,(x,y,z),dp;
sparsepoly(5);"";
↦ 24263xy4+24170x4z+21962x3yz+26642xy3z+5664xy2z2+17904xz4
↦
sparsepoly(3,5,90,9);
↦ 8x3z2+2y3z2+3xyz3+2xy3+yz3+xy2
```

D.2.10.8 sparsetriag

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparsetriag(n,m[,p,b]); n,m,p,b integers

Return: nxm lower triagonal integer matrix, diagonal entries equal to 1, about p percent of lower diagonal entries are 0, the remaining are random integers >=1 and <= b; [defaults: (p,b) = (75,1)]

Example:

```
LIB "random.lib";
sparsetriag(5,7);"";
↦ 1,0,0,0,0,0,0,
↦ 0,1,0,0,0,0,0,
↦ 0,1,1,0,0,0,0,
↦ 0,0,0,1,0,0,0,
↦ 1,1,0,0,1,0,0
↦
sparsetriag(7,5,90);"";
↦ 1,0,0,0,0,
↦ 0,1,0,0,0,
↦ 0,1,1,0,0,
↦ 0,0,0,1,0,
↦ 0,0,0,0,1,
↦ 0,0,0,1,0,
↦ 0,1,0,0,0
↦
sparsetriag(5,5,0);"";
↦ 1,0,0,0,0,
↦ 1,1,0,0,0,
↦ 1,1,1,0,0,
↦ 1,1,1,1,0,
↦ 1,1,1,1,1
↦
sparsetriag(5,5,50,100);
↦ 1,0,0,0,0,
↦ 73,1,0,0,0,
↦ 0,79,1,0,0,
↦ 14,0,0,1,0,
↦ 0,48,23,0,1
```

D.2.10.9 sparseHomogIdeal

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: sparseid(k,u[,o,p,b]); k,u,o,p,b integers

Return: ideal having k homogeneous generators, each of random degree in the interval [u,o], p percent of terms in degree d are 0, the remaining have random coefficients in the interval [1,b], (default: o=u, p=75, b=30000)

Example:

```
LIB "random.lib";
ring r = 0,(a,b,c,d),dp;
sparseHomogIdeal(2,3);"";
↦ _[1]=24004a3+12773a2b+6204a2c+20030b2c+19505bcd
↦ _[2]=817b3+9650c3+28857c2d+7247bd2+22567cd2
↦
sparseHomogIdeal(3,0,4,90,9);
↦ _[1]=5d
↦ _[2]=abc2+4ab2d+c3d+c2d2
↦ _[3]=3a
```

D.2.10.10 triagmatrix

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: triagmatrix(n,m,o[,u,pe,pp,b]); n,m,o,u,pe,pp,b integers

Return: nxm lower triagonal matrix, diagonal entries equal to 1, about p percent of lower diagonal entries are 0, the remaining are random polynomials of degree d, u<=d<=o, with pp percent of the terms being 0, the remaining have random coefficients in the interval [1,b] [default: (pe,u,pp,b) = (0,50,75,100)]

Example:

```
LIB "random.lib";
ring r = 0,(a,b,c,d),dp;
// sparse triagonal matrix of sparse polys of degree <=2:
print(triagmatrix(3,4,2));"";
↦ 1,                                 0,0,0,
↦ 52ac+54cd+14c,                     1,0,0,
↦ 17a2+19b2+45ac+94bc+50b+87c+54d+21,0,1,0
↦ 
// dense triagonal matrix of sparse linear forms:
print(triagmatrix(3,3,1,1,0,55,9));
↦ 1,       0,    0,
↦ 7a+8d,   1,    0,
↦ 9b+7c+4d,7b+9d,1
```

D.2.10.11 randomLast

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: randomLast(b); b int

Return: ideal = maxideal(1), but the last variable is exchanged by a random linear combination of all variables, with coefficients in the interval [-b,b], except for the last variable which always has coefficient 1

Example:

```
LIB "random.lib";
ring  r = 0,(x,y,z),lp;
ideal i = randomLast(10);
i;
↦ i[1]=x
↦ i[2]=y
↦ i[3]=-x+z
```

D.2.10.12 randomBinomial

Procedure from library `random.lib` (see Section D.2.10 [random_lib], page 935).

Usage: randomBinomial(k,u[,o,b]); k,u,o,b integers

Return: binomial ideal, k homogeneous generators of degree d, u<=d<=o, with randomly chosen monomials and coefficients in the interval [-b,b] (default: u=o, b=10).

Example:

```
LIB "random.lib";
ring  r = 0,(x,y,z),lp;
ideal i = randomBinomial(4,5,6);
i;
↦ i[1]=-x4z-xz4
↦ i[2]=8x2y3+8xy3z
↦ i[3]=-4x2y2z2-4xy5
↦ i[4]=5x3yz2+5xz5
```

D.2.11 resources_lib

Library: resources.lib

Purpose: Tools to manage the computational resources

Author: Andreas Steenpass, e-mail: steenpass@mathematik.uni-kl.de

Overview: The purpose of this library is to manage the computational resources of a Singular session. The library tasks.lib and any library build upon tasks.lib respect these settings, i.e. they will not use more computational resources than provided via resources.lib.

The provided procedures and their implementation are currently quite simple. The library can be extended later on to support, e.g., distributed computations on several servers.

Procedures: See also: Section D.2.7 [parallel_lib], page 884; Section D.2.13 [tasks_lib], page 958.

D.2.11.1 addcores

Procedure from library `resources.lib` (see Section D.2.11 [resources_lib], page 941).

Usage: addcores(n), n int

Return: the adjusted number of available processor cores, after n has been added to it. If n is negative, this number is reduced.

Note: The number of available processor cores must be at least 1. Reducing this number may take some time.
This procedure should only be called in the main process of a Singular session and not within any task defined via tasks.lib.

Example:

```
LIB "resources.lib";
setcores(4);
↦ 4
addcores(-2);
↦ 2
```

See also: Section D.2.11.3 [getcores], page 942; Section D.2.7 [parallel_lib], page 884; Section D.2.11.2 [setcores], page 941; Section D.2.13 [tasks_lib], page 958.

D.2.11.2 setcores

Procedure from library `resources.lib` (see Section D.2.11 [resources_lib], page 941).

Usage: setcores(n), n int

Return: n. The number of available processor cores is set to n and n is returned.

Note: The number of available processor cores must be at least 1. Reducing this number may take some time.
This procedure should only be called in the main process of a Singular session and not within any task defined via tasks.lib.

Example:

```
LIB "resources.lib";
setcores(2);
↦ 2
```

```
setcores(4);
↦ 4
```

See also: Section D.2.11.1 [addcores], page 941; Section D.2.11.3 [getcores], page 942; Section D.2.7 [parallel_lib], page 884; Section D.2.13 [tasks_lib], page 958.

D.2.11.3 getcores

Procedure from library `resources.lib` (see Section D.2.11 [resources_lib], page 941).

Usage: getcores(n), n int

Return: the number of available processor cores.

Note: This procedure should only be called in the main process of a Singular session and not within any task defined via tasks.lib.

Example:

```
LIB "resources.lib";
setcores(4);
↦ 4
getcores();
↦ 4
```

See also: Section D.2.11.1 [addcores], page 941; Section D.2.7 [parallel_lib], page 884; Section D.2.11.2 [setcores], page 941; Section D.2.13 [tasks_lib], page 958.

D.2.11.4 semaphore

Procedure from library `resources.lib` (see Section D.2.11 [resources_lib], page 941).

Usage: semaphore(n), n int

Return: the index of a new semaphore initialized with n.

Example:

```
LIB "resources.lib";
int sem = semaphore(1);
system("semaphore", "acquire", sem);
↦ 1
system("semaphore", "try_acquire", sem);
↦ 0
system("semaphore", "release", sem);
↦ 1
system("semaphore", "try_acquire", sem);
↦ 1
```

D.2.12 ring_lib

Library: ring.lib

Purpose: Manipulating Rings and Maps

Authors: Singular team

Procedures:

D.2.12.1 changechar

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: changechar(c[,r]); c=list, r=ring

Return: ring R, obtained from the ring r [default: r=basering], by changing ring_list(r)[1] to c.

Example:

```
LIB "ring.lib";
ring rr=2,A,dp;
ring r=0,(x,y,u,v),(dp(2),ds);
def R=changechar(ringlist(rr)); R;"";
↦ // coefficients: ZZ/2(A)
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering ds
↦ //                  : names    u v
↦ //        block   3 : ordering C
↦
def R1=changechar(32003,R); setring R1; R1;
↦ // coefficients: ZZ/32003
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering ds
↦ //                  : names    u v
↦ //        block   3 : ordering C
kill R,R1;
```

D.2.12.2 changeord

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: changeord(neword[,r]); newordstr=list, r=ring/qring

Return: ring R, obtained from the ring r [default: r=basering], by changing order(r) to neword. If, say, neword=list(list("wp",intvec(2,3)),list(list("dp",1:(n-2)))); and if the ring r exists and has n variables, the ring R will be equipped with the monomial ordering wp(2,3),dp.

Example:

```
LIB "ring.lib";
ring r=0,(x,y,u,v),(dp(2),ds);
def R=changeord(list(list("wp",intvec(2,3)),list("dp",1:2))); R; "";
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering wp
↦ //                  : names    x y
↦ //                  : weights  2 3
↦ //        block   2 : ordering dp
↦ //                  : names    u v
↦ //        block   3 : ordering C
↦
ideal i = x^2,y^2-u^3,v;
```

```
qring Q = std(i);
def Q'=changeord(list(list("lp",nvars(Q))),Q); setring Q'; Q';
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    x y u v
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=v
↦ _[2]=x2
↦ _[3]=y2-u3
kill R,Q,Q';
```

D.2.12.3 changevar

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: changevar(vars[,r]); vars=string, r=ring/qring

Return: ring R, obtained from the ring r [default: r=basering], by changing varstr(r) according to the value of vars.
If, say, vars = "t()" and the ring r exists and has n variables, the new basering will have name R and variables t(1),...,t(n).
If vars = "a,b,c,d", the new ring will have the variables a,b,c,d.

Note: This procedure is useful in connection with the procedure ringtensor, when a conflict between variable names must be avoided. This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
ring r=0,(x,y,u,v),(dp(2),ds);
ideal i = x^2,y^2-u^3,v;
qring Q = std(i);
setring(r);
def R=changevar("A()"); R; "";
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    A(1) A(2)
↦ //        block   2 : ordering ds
↦ //                  : names    A(3) A(4)
↦ //        block   3 : ordering C
↦
def Q'=changevar("a,b,c,d",Q); setring Q'; Q';
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    a b
↦ //        block   2 : ordering ds
↦ //                  : names    c d
↦ //        block   3 : ordering C
↦ // quotient ring from ideal
↦ _[1]=d
↦ _[2]=a2
↦ _[3]=b2-c3
kill R,Q,Q';
```

D.2.12.4 defring

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: defring(ch,n,va,or); ch,va,or=strings, n=integer

Return: ring R with characteristic 'ch', ordering 'or' and n variables with names derived from va.
If va is a single letter, say va="a", and if n<=26 then a and the following n-1 letters from the alphabet (cyclic order) are taken as variables. If n>26 or if va is a single letter followed by a bracket, say va="T(", the variables are T(1),...,T(n).

Note: This proc is useful for defining a ring in a procedure. This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
def r=defring("0",5,"u","ls"); r; setring r;"";
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering ls
↦ //                  : names    u v w x y
↦ //        block   2 : ordering C
↦
def R=defring("(2,A)",10,"x(","(dp(3),ws(1,2,3),ds)"); R; setring R;
↦ // coefficients: ZZ/2(A)
↦ // number of vars : 10
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering ws
↦ //                  : names    x(4) x(5) x(6)
↦ //                  : weights     1    2    3
↦ //        block   3 : ordering ds
↦ //                  : names    x(7) x(8) x(9) x(10)
↦ //        block   4 : ordering C
```

D.2.12.5 defrings

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: defrings(n,[p]); n,p integers

Return: ring R with characteristic p [default: p=32003], ordering ds and n variables x,y,z,a,b,...if n<=26 (resp. x(1..n) if n>26)

Note: This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
def S5=defrings(5,0); S5; "";
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering ds
↦ //                  : names    x y z a b
↦ //        block   2 : ordering C
↦
def S30=defrings(30); S30;
↦ // coefficients: ZZ/32003
↦ // number of vars : 30
↦ //        block   1 : ordering ds
↦ //                  : names    x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(\
   9) x(10) x(11) x(12) x(13) x(14) x(15) x(16) x(17) x(18) x(19) x(20) x(21\
   ) x(22) x(23) x(24) x(25) x(26) x(27) x(28) x(29) x(30)
↦ //        block   2 : ordering C
kill S5,S30;
```

D.2.12.6 defringp

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: defringp(n,[p]); n,p=integers

Return: ring R with characteristic p [default: p=32003], ordering dp and n variables x,y,z,a,b,...if n<=26 (resp. x(1..n) if n>26)

Note: This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
def P5=defringp(5,0); P5; "";
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y z a b
↦ //        block   2 : ordering C
↦
def P30=defringp(30); P30;
↦ // coefficients: ZZ/32003
↦ // number of vars : 30
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3) x(4) x(5) x(6) x(7) x(8) x(\
   9) x(10) x(11) x(12) x(13) x(14) x(15) x(16) x(17) x(18) x(19) x(20) x(21\
   ) x(22) x(23) x(24) x(25) x(26) x(27) x(28) x(29) x(30)
↦ //        block   2 : ordering C
kill P5,P30;
```

D.2.12.7 extendring

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: extendring(n,va,o[,iv,i,r]); va,o=strings, n,i=integers, r=ring, iv=intvec of positive integers or iv=0

Return: ring R, which extends the ring r by adding n new variables in front of (resp. after, if i!=0) the old variables.
[default: (i,r)=(0,basering)].
– The characteristic is the characteristic of r.
– The new vars are derived from va. If va is a single letter, say va="T", and if n<=26

then T and the following n-1 letters from T..Z..T (resp. T(1..n) if n>26) are taken as additional variables. If va is a single letter followed by a bracket, say va="x(", the new variables are x(1),...,x(n).
– The ordering is the product ordering of the ordering of r and of an ordering derived from ‘o‘ [and iv].
- If o contains a ’c’ or a ’C’ in front resp. at the end, this is taken for the whole ordering in front, resp. at the end. If o does not contain a ’c’ or a ’C’ the same rule applies to ordstr(r).
- If no intvec iv is given, or if iv=0, o may be any allowed ordstr, like "ds" or "dp(2),wp(1,2,3),Ds(2)" or "ds(a),dp(b),ls" if a and b are globally (!) defined integers and if a+b+1<=n. If, however, a and b are local to a proc calling extendring, the intvec iv must be used to let extendring know the values of a and b
- If a non-zero intvec iv is given, iv[1],iv[2],... are taken for the 1st, 2nd,... block of o, if o contains no substring "w" or "W" i.e. no weighted ordering (in the above case o="ds,dp,ls" and iv=a,b).
If o contains a weighted ordering (only one (!) weighted block is allowed) iv[1] is taken as size for the weight-vector, the next iv[1] values of iv are taken as weights and the remaining values of iv as block size for the remaining non-weighted blocks. e.g. o="dp,ws,Dp,ds", iv=3,2,3,4,2,5 creates the ordering dp(2),ws(2,3,4),Dp(5),ds

Note: This proc is useful for adding deformation parameters.
This proc uses ’execute’ or calls a procedure using ’execute’. If you use it in your own proc, it may be advisable to let the local names of your proc start with a @

Example:

```
LIB "ring.lib";
ring r=0,(x,y,z),ds;
show(r);"";
↦ // ring: (QQ),(x,y,z),(ds(3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦
// blocksize is derived from no of vars:
int t=5;
def R1=extendring(t,"a","dp");         //t global: "dp" -> "dp(5)"
show(R1); setring R1; "";
↦ // ring: (QQ),(a,b,c,d,e,x,y,z),(dp(5),ds(3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦
def R2=extendring(4,"T(","c,dp",1,r);    //"dp" -> "c,..,dp(4)"
show(R2); setring R2; "";
↦ // ring: (QQ),(x,y,z,T(1),T(2),T(3),T(4)),(c,ds(3),dp(4));
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦
// no intvec given, blocksize given: given blocksize is used:
def R3=extendring(4,"T(","dp(2)",0,r);   // "dp(2)" -> "dp(2)"
show(R3); setring R3; "";
↦ // ring: (QQ),(T(1),T(2),T(3),T(4),x,y,z),(dp(2),ds(5),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦
```

```
// intvec given: weights and blocksize is derived from given intvec
// (no specification of a blocksize in the given ordstr is allowed!)
// if intvec does not cover all given blocks, the last block is used
// for the remaining variables, if intvec has too many components,
// the last ones are ignored
intvec v=3,2,3,4,1,3;
def R4=extendring(10,"A","ds,ws,Dp,dp",v,0,r);
// v covers 3 blocks: v[1] (=3) : no of components of ws
// next v[1] values (=v[2..4]) give weights
// remaining components of v are used for the remaining blocks
show(R4);
↦ // ring: (QQ),(A,B,C,D,E,F,G,H,I,J,x,y,z),(ds(1),ws(2,3,4),Dp(3),dp(3),ds\
   (3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
kill r,R1,R2,R3,R4;
```

D.2.12.8 fetchall

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: fetchall(R[,s]); R=ring/qring, s=string

Create: fetch all objects of ring R (of type poly/ideal/vector/module/number/matrix) into the basering.
If no 2nd argument is present, the names are the same as in R. If, say, f is a polynomial in R and the 2nd argument is the string "R", then f is mapped to f_R etc.

Return: no return value

Note: As fetch, this procedure maps the 1st, 2nd, ... variable of R to the 1st, 2nd, ... variable of the basering.
The 2nd argument is useful in order to avoid conflicts of names, the empty string is allowed

Caution: fetchall does not work for locally defined names.
It does not work if R contains a map.

Example:

```
LIB "ring.lib";
// The example is not shown since fetchall does not work in a procedure;
// (and hence not in the example procedure). Try the following commands:
//   ring R=0,(x,y,z),dp;
//   ideal j=x,y2,z2;
//   matrix M[2][3]=1,2,3,x,y,z;
//   j; print(M);
//   ring S=0,(a,b,c),ds;
//   fetchall(R);       //map from R to S: x->a, y->b, z->c;
//   names(S);
//   j; print(M);
//   fetchall(S,"1");   //identity map of S: copy objects, change names
//   names(S);
//   kill R,S;
```

See also: Section D.2.12.9 [imapall], page 949.

D.2.12.9 imapall

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: imapall(R[,s]); R=ring/qring, s=string

Create: map all objects of ring R (of type poly/ideal/vector/module/number/matrix) into the basering by applying imap to all objects of R. If no 2nd argument is present, the names are the same as in R. If, say, f is a polynomial in R and the 3rd argument is the string "R", then f is mapped to f_R etc.

Return: no return value

Note: As imap, this procedure maps the variables of R to the variables with the same name in the basering, the other variables are mapped to 0. The 2nd argument is useful in order to avoid conflicts of names, the empty string is allowed

Caution: imapall does not work for locally defined names.
It does not work if R contains a map

Example:

```
LIB "ring.lib";
// The example is not shown since imapall does not work in a procedure
// (and hence not in the example procedure). Try the following commands:
//   ring R=0,(x,y,z,u),dp;
//   ideal j=x,y,z,u2+ux+z;
//   matrix M[2][3]=1,2,3,x,y,uz;
//   j; print(M);
//   ring S=0,(a,b,c,x,z,y),ds;
//   imapall(R);         //map from R to S: x->x, y->y, z->z, u->0
//   names(S);
//   j; print(M);
//   imapall(S,"1");     //identity map of S: copy objects, change names
//   names(S);
//   kill R,S;
```

See also: Section D.2.12.8 [fetchall], page 948.

D.2.12.10 mapall

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: mapall(R,i[,s]); R=ring/qring, i=ideal of basering, s=string

Create: map all objects of ring R (of type poly/ideal/vector/module/number/ matrix, map) into the basering by mapping the j-th variable of R to the j-th generator of the ideal i. If no 3rd argument is present, the names are the same as in R. If, say, f is a polynomial in R and the 3rd argument is the string "R", then f is mapped to f_R etc.

Return: no return value.

Note: This procedure has the same effect as defining a map, say psi, by map psi=R,i; and then applying psi to all objects of R. In particular, maps from R to some ring S are composed with psi, creating thus a map from the basering to S.
mapall may be combined with copyring to change vars for all objects. The 3rd argument is useful in order to avoid conflicts of names, the empty string is allowed.

Caution: mapall does not work for locally defined names.

Example:

```
LIB "ring.lib";
// The example is not shown since mapall does not work in a procedure
// (and hence not in the example procedure). Try the following commands:
//   ring R=0,(x,y,z),dp;
//   ideal j=x,y,z;
//   matrix M[2][3]=1,2,3,x,y,z;
//   map phi=R,x2,y2,z2;
//   ring S=0,(a,b,c),ds;
//   ideal i=c,a,b;
//   mapall(R,i);         //map from R to S: x->c, y->a, z->b
//   names(S);
//   j; print(M); phi;    //phi maps R to S: x->c2, y->a2, z->b2
//   ideal i1=a2,a+b,1;
//   mapall(R,i1,"");     //map from R to S: x->a2, y->a+b, z->1
//   names(S);
//   j_; print(M_); phi_;
//   changevar("T","x()",R);  //change vars in R and call result T
//   mapall(R,maxideal(1));   //identity map from R to T
//   names(T);
//   j; print(M); phi;
//   kill R,S,T;
```

D.2.12.11 ord_test

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: ord_test(r); r ring/qring

Return: int 1 (resp. -1, resp. 0) if ordering of r is global (resp. local, resp. mixed)

Example:

```
LIB "ring.lib";
ring R = 0,(x,y),dp;
ring S = 0,(u,v),ls;
ord_test(R);
↦ 1
ord_test(S);
↦ -1
ord_test(R+S);
↦ 0
```

See also: Section 5.1.2 [attrib], page 156.

D.2.12.12 ringtensor

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: ringtensor(r1,r2,...); r1,r2,...=rings

Return: ring R whose variables are the variables from all rings r1,r2,... and whose monomial ordering is the block (product) ordering of the respective monomial orderings of r1,r2,... . Hence, R is the tensor product of the rings r1,r2,... with ordering matrix equal to the direct sum of the ordering matrices of r1,r2,...

Note: The characteristic of the new ring will be p if one ring has characteristic p. The names of variables in the rings r1,r2,... must differ.
The procedure works also for quotient rings ri, if the characteristic of ri is compatible with the characteristic of the result (i.e. if imap from ri to the result is implemented)

Example:

```
LIB "ring.lib";
ring r=32003,(x,y,u,v),dp;
ring s=0,(a,b,c),wp(1,2,3);
ring t=0,x(1..5),(c,ls);
def R=ringtensor(r,s,t);
type R;
↦ // R ring
↦ // coefficients: ZZ/32003
↦ // number of vars : 12
↦ //        block   1 : ordering dp
↦ //                  : names    x y u v
↦ //        block   2 : ordering wp
↦ //                  : names    a b c
↦ //                  : weights  1 2 3
↦ //        block   3 : ordering ls
↦ //                  : names    x(1) x(2) x(3) x(4) x(5)
↦ //        block   4 : ordering C
setring s;
ideal i = a2+b3+c5;
def S=changevar("x,y,z");       //change vars of s
setring S;
qring qS =std(fetch(s,i));      //create qring of S mod i (mapped to S)
def T=changevar("d,e,f,g,h",t); //change vars of t
setring T;
qring qT=std(d2+e2-f3);         //create qring of T mod d2+e2-f3
def Q=ringtensor(s,qS,t,qT);
setring Q; type Q;
↦ // Q ring
↦ // coefficients: QQ
↦ // number of vars : 16
↦ //        block   1 : ordering wp
↦ //                  : names    a b c
↦ //                  : weights  1 2 3
↦ //        block   2 : ordering wp
↦ //                  : names    x y z
↦ //                  : weights  1 2 3
↦ //        block   3 : ordering ls
↦ //                  : names    x(1) x(2) x(3) x(4) x(5)
↦ //        block   4 : ordering ls
↦ //                  : names    d e f g h
↦ //        block   5 : ordering C
↦ // quotient ring from ideal
↦ _[1]=z5+y3+x2
↦ _[2]=f3-e2-d2
kill R,S,T,Q;
```

See also: Section 4.20.4 [ring operations], page 128.

D.2.12.13 ringweights

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: ringweights(P); P=name of an existing ring (true name, not a string)

Return: intvec consisting of the weights of the variables of P, as they appear when typing P;.

Note: This is useful when enlarging P but keeping the weights of the old variables.

Example:

```
LIB "ring.lib";
ring r0 = 0,(x,y,z),dp;
ringweights(r0);
↦ 1,1,1
ring r1 = 0,x(1..5),(ds(3),wp(2,3));
ringweights(r1);"";
↦ 1,1,1,2,3
↦
// an example for enlarging the ring, keeping the first weights:
intvec v = ringweights(r1),6,2,3,4,5;
ring R = 0,x(1..10),(a(v),dp);
ordstr(R);
↦ a(1,1,1,2,3,6,2,3,4,5),dp(10),C
```

D.2.12.14 preimageLoc

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: preimageLoc (ring_name, map_name, ideal_name);
all input parameters of type string

Return: ideal

Purpose: compute the preimage of an ideal under a given map for non-global orderings.
The 2nd argument has to be the name of a map from the basering to the given ring (or the name of an ideal defining such a map), and the ideal has to be an ideal in the given ring.

Example:

```
LIB "ring.lib";
ring S =0,(x,y,z),dp;
ring R0=0,(x,y,z),ds;
qring R=std(x+x2); if(voice>1) {export R;}
map psi=S,x,y,z;   if(voice>1) {export psi;}
ideal null;        if(voice>1) {export null;}
setring S;
ideal nu=preimageLoc("R","psi","null");
nu;
↦ nu[1]=x
```

See also: Section 5.1.117 [preimage], page 240.

D.2.12.15 rootofUnity

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: rootofUnity(n); n an integer

Return: number

Purpose: compute the minimal polynomial for the n-th primitive root of unity

Note: works only in field extensions by one element

Example:

```
LIB "ring.lib";
ring r = (0,q),(x,y,z),dp;
rootofUnity(6);
↦ (q2-q+1)
rootofUnity(7);
↦ (q6+q5+q4+q3+q2+q+1)
minpoly = rootofUnity(8);
r;
↦ // coefficients: QQ[q]/(q4+1)
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

D.2.12.16 optionIsSet

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: optionIsSet(optionName)

Parameters:
optionName: a name as string of an option of interest

Return: true, if the by optionName given option is active, false otherwise.

Example:

```
LIB "ring.lib";
// check if the option "warn" is set.
optionIsSet("warn");
↦ 0
option("warn");
// now the option is set
optionIsSet("warn");
↦ 1
option("nowarn");
// now the option is unset
optionIsSet("warn");
↦ 0
```

D.2.12.17 hasNumericCoeffs

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasNumericCoeffs (rng);

Return: 1 if rng has inexact coefficcients, 0 otherwise.

Example:

```
LIB "ring.lib";
ring r1 = integer,x,dp;
hasNumericCoeffs(r1);
↦ 0
ring r2 = complex,x,dp;
hasNumericCoeffs(r2);
↦ 1
```

D.2.12.18 hasCommutativeVars

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasCommutativeVars (rng);

Return: 1 if rng is a commutative polynomial ring, 0 otherwise.

Example:

```
LIB "ring.lib";
ring r=0,(x,y,z),dp;
hasCommutativeVars(r);
↦ 1
```

D.2.12.19 hasGlobalOrdering

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasGlobalOrdering (rng);

Return: 1 if rng has a global monomial ordering, 0 otherwise.

Example:

```
LIB "ring.lib";
ring rng = integer,x,dp;
hasGlobalOrdering(rng); //yes
↦ 1
ring rng2 = 0, x, ds;
hasGlobalOrdering(rng2);  // no
↦ 0
```

D.2.12.20 hasMixedOrdering

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasMixedOrdering();

Return: 1 if ordering of basering is mixed, 0 else

Example:

```
LIB "ring.lib";
ring R1 = 0,(x,y,z),dp;
hasMixedOrdering();
↦ 0
ring R2 = 31,(x(1..4),y(1..3)),(ds(4),lp(3));
hasMixedOrdering();
↦ 1
ring R3 = 181,x(1..9),(dp(5),lp(4));
hasMixedOrdering();
↦ 0
```

D.2.12.21 hasAlgExtensionCoefficient

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasAlgExtensionCoefficient (rng);

Return: 1 if the coefficients are an algebraic extension, 0 otherwise.

Example:

```
LIB "ring.lib";
ring rng = integer,x,dp;
hasAlgExtensionCoefficient(rng); //no
↦ 0
ring rng2 = (0,a), x, dp; minpoly=a2-1;
hasAlgExtensionCoefficient(rng2);  // yes
↦ 1
ring rng3=(49,a),x,dp;
hasAlgExtensionCoefficient(rng3);  // no
↦ 0
```

D.2.12.22 hasTransExtensionCoefficient

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: hasTransExtensionCoefficient (rng);

Return: 1 if the coefficients are rational functions, 0 otherwise.

Example:

```
LIB "ring.lib";
ring rng = integer,x,dp;
hasTransExtensionCoefficient(rng); //no
↦ 0
ring rng2 = (0,a), x, dp;
hasTransExtensionCoefficient(rng2);  // yes
↦ 1
ring rng3=(49,a),x,dp;
hasTransExtensionCoefficient(rng3);  // no
↦ 0
```

D.2.12.23 isQuotientRing

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Return: 1 if rng is a quotient ring, 0 otherwise.

Purpose: check if typeof a rng "qring"

Example:

```
LIB "ring.lib";
ring rng = 0,x,dp;
isQuotientRing(rng); //no
↦ 0
// if a certain method does not support quotient rings,
// then a parameter test could be performed:
ASSUME( 0, 0==isQuotientRing(basering));
qring q= ideal(x);  // constructs rng/ideal(x)
isQuotientRing(q);  // yes
↦ 1
```

D.2.12.24 isSubModule

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Return: 1 if module(I) is in module(J), 0 otherwise

Example:

```
LIB "ring.lib";
ring r=0,x,dp;
ideal I1=x2;
ideal I2=x3;
isSubModule(I1, I2);
↦ 0
isSubModule(I2, I1);
↦ 1
```

D.2.12.25 changeordTo

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: changeordTo(ring, string s);

Return: a ring with the oderinging changed to the (simple) ordering s

Example:

```
LIB "ring.lib";
ring r=0,(x,y),lp;
def rr=changeordTo(r,"dp");
rr;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x y
```

D.2.12.26 addvarsTo

Procedure from library `ring.lib` (see Section D.2.12 [ring_lib], page 942).

Usage: addvarsTo(ring,list_of_strings, int);
int may be: 0:ordering: dp
1:ordering dp,dp
2:oring.ordering,dp

Return: a ring with the additional variables

Example:

```
LIB "ring.lib";
ring r=0,(x,y),lp;
def rr=addvarsTo(r,list("a","b"),0);
rr; kill rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x y a b
```

```
def rr=addvarsTo(r,list("a","b"),1);
rr; kill rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x y
↦ //        block   3 : ordering dp
↦ //                  : names    a b
def rr=addvarsTo(r,list("a","b"),2);
rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    x y
↦ //        block   2 : ordering C
↦ //        block   3 : ordering dp
↦ //                  : names    a b
```

D.2.12.27 addNvarsTo

Procedure from library ring.lib (see Section D.2.12 [ring_lib], page 942).

Usage: addNvarsTo(ring,int N, string name, int b);
b may be: 0:ordering: dp
1:ordering dp,dp
2:oring.ordering,dp

Return: a ring with N additional variables

Example:

```
LIB "ring.lib";
ring r=0,(x,y),lp;
def rr=addNvarsTo(r,2,"@",0);
rr; kill rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x y @(1) @(2)
def rr=addNvarsTo(r,2,"@",1);
rr; kill rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x y
↦ //        block   3 : ordering dp
↦ //                  : names    @(1) @(2)
def rr=addNvarsTo(r,2,"@",2);
rr;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering lp
↦ //                  : names    x y
↦ //        block   2 : ordering C
↦ //        block   3 : ordering dp
↦ //                  : names    @(1) @(2)
```

D.2.13 tasks_lib

Library: tasks.lib

Purpose: A parallel framework based on tasks

Author: Andreas Steenpass, e-mail: steenpass@mathematik.uni-kl.de

Overview: This library provides a parallel framework based on tasks. It introduces a new Singular type `task`; an object of this type is a command (given by a string) applied to a list of arguments. Tasks can be computed in parallel via the procedures in this library and they can even be started recursively, i.e. from within other tasks.

tasks.lib respects the limits for computational resources defined in Section D.2.11 [resources_lib], page 941, i.e., all tasks within the same Singular session will not use more computational resources than provided via resources.lib, even if tasks are started recursively.

The Singular library Section D.2.7 [parallel_lib], page 884 provides implementations of several parallel 'skeletons' based on tasks.lib.

Procedures: See also: Section D.2.7 [parallel_lib], page 884; Section D.2.11 [resources_lib], page 941.

D.2.13.1 createTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: createTask(command, arguments), command string, arguments list

Return: a task with the given command and arguments whose state is 'created'.

Note: 't = command, arguments;' is a shortcut for
't = createTask(command, arguments);'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = createTask("std", list(I));
// This is the same as:
// task t = "std", list(I);
t;
↦ A task with the following properties:
↦ command:          std
↦ no. of arguments: 1
↦ state:            created
↦
killTask(t);
```

See also: Section D.2.13.4 [compareTasks], page 960; Section D.2.13.3 [copyTask], page 959; Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.14 [getState], page 967; Section D.2.13.2 [killTask], page 959; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961.

D.2.13.2 killTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: killTask(t), t task

Return: nothing. If the state of t is 'started', then t is stopped first. The internal data structures of t are erased and its state is set to 'uninitialized'.

Note: 'killTask(t);' is not the same as 'kill t;'. The latter command does not erase the internal data structures of t. Hence killTask() should be called for any no longer needed task in order to free memory.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
startTasks(t);
t;
↦ A task with the following properties:
↦ command:          std
↦ no. of arguments: 1
↦ state:            started
↦
killTask(t);
t;
↦ An uninitialized task
↦
getState(t);
↦ uninitialized
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.14 [getState], page 967; Section D.2.13.5 [printTask], page 961; Section D.2.13.7 [stopTask], page 962.

D.2.13.3 copyTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: copyTask(t), t task

Return: a copy of t.

Note: 'task t1 = copyTask(t2);' is not the same as 'task t1 = t2;'. After the latter command, t1 points to the same object as t2; any changes to t2 will also effect t1. In contrast to this, copyTask() creates a new independent task.
A task whose state is 'started' cannot be copied.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t1 = "std", list(I);
startTasks(t1);
waitAllTasks(t1);
task t2 = copyTask(t1);
killTask(t1);
```

```
t2;   // t2 survived
↦ A task with the following properties:
↦ command:          std
↦ no. of arguments: 1
↦ state:            completed
↦
getResult(t2);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
killTask(t2);
```

See also: Section D.2.13.4 [compareTasks], page 960; Section D.2.13.1 [createTask], page 958; Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.2 [killTask], page 959; Section D.2.13.5 [printTask], page 961.

D.2.13.4 compareTasks

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: compareTasks(t1, t2), t1, t2 tasks

Return: 1, if t1 and t2 coincide;
0, otherwise.

Note: The arguments and the results of t1 and t2 are not compared.
't1 == t2' is a shortcut for 'compareTasks(t1, t2)'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t1 = "std", list(I);
task t2 = "std", list(I);
compareTasks(t1, t2);
↦ 1
startTasks(t1);
waitAllTasks(t1);
t1 == t2;   // the same as compareTasks(t1, t2);
↦ 0
killTask(t1);
killTask(t2);
// The arguments and the result are not compared!
ideal J = x;
task t3 = "std", list(I);
task t4 = "std", list(J);
t3 == t4;
↦ 1
killTask(t3);
killTask(t4);
```

See also: Section D.2.13.3 [copyTask], page 959; Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.5 [printTask], page 961.

D.2.13.5 printTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: printTask(t), t task

Return: nothing. Prints information about t.

Note: 'print(t);' and 't;' are shortcuts for 'printTask(t)'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t;
printTask(t);
↦ An uninitialized task
t = "std", list(I);
t;   // the same as printTask(t);
↦ A task with the following properties:
↦ command:           std
↦ no. of arguments: 1
↦ state:             created
↦
startTasks(t);
waitAllTasks(t);
t;
↦ A task with the following properties:
↦ command:           std
↦ no. of arguments: 1
↦ state:             completed
↦
killTask(t);
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.2 [killTask], page 959.

D.2.13.6 startTasks

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: startTasks(t1, t2, ...), t1, t2, ... tasks

Return: nothing. Starts the tasks t1, t2, ... and sets their states to 'started'.

Note: A task whose state is neither 'created' nor 'stopped' cannot be started.
If startTasks() is applied to a task whose state is 'stopped', then the computation of this task will be restarted from the beginning.
Tasks can be started from within other tasks. A started task should not be accessed from within any task other than the one within which it was started.
For each task, the start of its computation is subject to the internal scheduling.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t1 = "std", list(I);
task t2 = "slimgb", list(I);
startTasks(t1, t2);
waitAllTasks(t1, t2);
getResult(t1);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
getResult(t2);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=xy13+y12
↦ _[6]=y14+xy12
killTask(t1);
killTask(t2);
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.14 [getState], page 967; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.7 [stopTask], page 962; Section D.2.13.8 [waitTasks], page 963.

D.2.13.7 stopTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: stopTask(t), t task

Return: nothing. Stops the t and sets its state to 'stopped'.

Note: A task whose state is not 'started' cannot be stopped.
Intermediate results are discarded when a task is stopped.
killTask() should be called for any no longer needed task.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
startTasks(t);
stopTask(t);
t;
↦ A task with the following properties:
↦ command:          std
↦ no. of arguments: 1
↦ state:            stopped
↦
killTask(t);
```

See also: Section D.2.13.14 [getState], page 967; Section D.2.13.2 [killTask], page 959; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961; Section D.2.13.8 [waitTasks], page 963.

D.2.13.8 waitTasks

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: waitTasks(T, N[, timeout]), T list of tasks, N int, timeout int

Return: an ordered list of the indices of those tasks which have been successfully completed. The state of these tasks is set to 'completed'.
The procedure waits for N tasks to complete.
An optional timeout in ms can be provided. Default is 0 which disables the timeout.

Note: A task whose state is neither 'started' nor 'completed' cannot be waited for.
The result of any completed task can be accessed via Section D.2.13.13 [getResult], page 966.
The returned list may contain more than N entries if the computation of some tasks has already finished and/or if several tasks finish "at the same time". It may contain less than N entries in the case of timeout or errors occurring.
Polling is guaranteed, i.e. the index of any task t for which 'pollTask(t);' would return 1 will appear in the returned list.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t1 = "std", list(I);
task t2 = "slimgb", list(I);
startTasks(t1, t2);
waitTasks(list(t1, t2), 2);   // wait for both tasks
↦ [1]:
↦    1
↦ [2]:
↦    2
getResult(t1);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
getResult(t2);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=xy13+y12
↦ _[6]=y14+xy12
killTask(t1);
killTask(t2);
```

See also: Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961.

D.2.13.9 waitAllTasks

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: waitAllTasks(t1, t2, ...), t1, t2, ... tasks

Return: nothing. Waits for all the tasks t1, t2, ... to complete. The state of the tasks is set to 'completed'.

Note: A task whose state is neither 'started' nor 'completed' cannot be waited for.
The result of any completed task can be accessed via Section D.2.13.13 [getResult], page 966.
'waitAllTasks(t1, t2, ...);' is a shortcut for 'waitTasks(list(t1, t2, ...), size(list(t1, t2, ...)));'. Since returning a list of the indices of the completed tasks does not make sense in this case, nothing is returned.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t1 = "std", list(I);
task t2 = "slimgb", list(I);
startTasks(t1, t2);
waitAllTasks(t1, t2);   // the same as 'waitTasks(list(t1, t2), 2);',
// but without return value
getResult(t1);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
getResult(t2);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=xy13+y12
↦ _[6]=y14+xy12
killTask(t1);
killTask(t2);
```

See also: Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961; Section D.2.13.8 [waitTasks], page 963.

D.2.13.10 pollTask

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: pollTask(t), t task

Return: 1, if the computation of the task t has successfully finished; 0, otherwise.
The state of any task whose computation has successfully finished is set to 'completed'.

Note: A task whose state is neither 'started' nor 'completed' cannot be polled.
The result of any completed task can be accessed via Section D.2.13.13 [getResult], page 966.
pollTask() should return immediately. However, receiving the result of the task may take some time.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
startTasks(t);
waitAllTasks(t);
pollTask(t);   // task already completed
↦ 1
t;
↦ A task with the following properties:
↦ command:          std
↦ no. of arguments: 1
↦ state:            completed
↦
getResult(t);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
killTask(t);
```

See also: Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961; Section D.2.13.8 [waitTasks], page 963.

D.2.13.11 getCommand

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: getCommand(t), t task

Return: a string, the command of t.

Note: This command cannot be applied to tasks whose state is
'uninitialized'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
getCommand(t);
↦ std
killTask(t);
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.12 [getArguments], page 966; Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.5 [printTask], page 961.

D.2.13.12 getArguments

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: getArguments(t), t task

Return: a list, the arguments of t.

Note: This command cannot be applied to tasks whose state is 'uninitialized'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
getArguments(t);
↦ [1]:
↦    _[1]=x9y2+x10
↦    _[2]=x2y7-y8
killTask(t);
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.11 [getCommand], page 965; Section D.2.13.13 [getResult], page 966; Section D.2.13.14 [getState], page 967; Section D.2.13.5 [printTask], page 961.

D.2.13.13 getResult

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: getResult(t), t task

Return: the result of t.

Note: This command cannot be applied to tasks whose state is not 'completed'.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
startTasks(t);
waitAllTasks(t);
getResult(t);
↦ _[1]=x2y7-y8
↦ _[2]=x9y2+x10
↦ _[3]=x12y+xy11
↦ _[4]=x13-xy12
↦ _[5]=y14+xy12
↦ _[6]=xy13+y12
killTask(t);
```

See also: Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.14 [getState], page 967; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.8 [waitTasks], page 963.

D.2.13.14 getState

Procedure from library `tasks.lib` (see Section D.2.13 [tasks_lib], page 958).

Usage: getState(t), t task

Return: a string, the state of t.

Example:

```
LIB "tasks.lib";
ring R = 0, (x,y), dp;
ideal I = x9y2+x10, x2y7-y8;
task t = "std", list(I);
getState(t);
↦ created
startTasks(t);
getState(t);
↦ started
waitAllTasks(t);
getState(t);
↦ completed
killTask(t);
getState(t);
↦ uninitialized
```

See also: Section D.2.13.1 [createTask], page 958; Section D.2.13.12 [getArguments], page 966; Section D.2.13.11 [getCommand], page 965; Section D.2.13.13 [getResult], page 966; Section D.2.13.2 [killTask], page 959; Section D.2.13.10 [pollTask], page 964; Section D.2.13.5 [printTask], page 961; Section D.2.13.6 [startTasks], page 961; Section D.2.13.7 [stopTask], page 962; Section D.2.13.8 [waitTasks], page 963.

D.3 Linear algebra

D.3.1 matrix_lib

Library: matrix.lib

Purpose: Elementary Matrix Operations

Procedures:

D.3.1.1 compress

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: compress(A); A matrix/ideal/module/intmat/intvec

Return: same type, zero columns/generators from A deleted
(if A=intvec, zero elements are deleted)

Example:

```
LIB "matrix.lib";
ring r=0,(x,y,z),ds;
matrix A[3][4]=1,0,3,0,x,0,z,0,x2,0,z2,0;
print(A);
↦ 1, 0,3, 0,
↦ x, 0,z, 0,
↦ x2,0,z2,0
print(compress(A));
↦ 1, 3,
↦ x, z,
↦ x2,z2
module m=module(A); show(m);
↦ // module, 4 generator(s)
↦ [1,x,x2]
↦ [0]
↦ [3,z,z2]
↦ [0]
show(compress(m));
↦ // module, 2 generator(s)
↦ [1,x,x2]
↦ [3,z,z2]
intmat B[3][4]=1,0,3,0,4,0,5,0,6,0,7,0;
compress(B);
↦ 1,3,
↦ 4,5,
↦ 6,7
intvec C=0,0,1,2,0,3;
compress(C);
↦ 1,2,3
```

D.3.1.2 concat

Procedure from library matrix.lib (see Section D.3.1 [matrix_lib], page 967).

Usage: concat(A1,A2,..); A1,A2,... matrices

Return: matrix, concatenation of A1,A2,.... Number of rows of result matrix is max(nrows(A1),nrows(A2),...)

Example:

```
LIB "matrix.lib";
ring r=0,(x,y,z),ds;
matrix A[3][3]=1,2,3,x,y,z,x2,y2,z2;
matrix B[2][2]=1,0,2,0; matrix C[1][4]=4,5,x,y;
print(A);
↦ 1, 2, 3,
↦ x, y, z,
↦ x2,y2,z2
print(B);
↦ 1,0,
↦ 2,0
print(C);
↦ 4,5,x,y
print(concat(A,B,C));
↦ 1, 2, 3, 1,0,4,5,x,y,
↦ x, y, z, 2,0,0,0,0,0,
↦ x2,y2,z2,0,0,0,0,0,0
```

D.3.1.3 diag

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: diag(p,n); p poly, n integer
diag(A); A matrix

Return: diag(p,n): diagonal matrix, p times unit matrix of size n.
diag(A) : n*m x n*m diagonal matrix with entries all the entries of the nxm matrix A, taken from the 1st row, 2nd row etc of A

Example:

```
LIB "matrix.lib";
ring r = 0,(x,y,z),ds;
print(diag(xy,4));
↦ xy,0, 0, 0,
↦ 0, xy,0, 0,
↦ 0, 0, xy,0,
↦ 0, 0, 0, xy
matrix A[3][2] = 1,2,3,4,5,6;
print(A);
↦ 1,2,
↦ 3,4,
↦ 5,6
print(diag(A));
↦ 1,0,0,0,0,0,
↦ 0,2,0,0,0,0,
↦ 0,0,3,0,0,0,
↦ 0,0,0,4,0,0,
↦ 0,0,0,0,5,0,
↦ 0,0,0,0,0,6
```

D.3.1.4 dsum

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: dsum(A1,A2,..); A1,A2,... matrices

Return: matrix, direct sum of A1,A2,...

Example:

```
LIB "matrix.lib";
ring r = 0,(x,y,z),ds;
matrix A[3][3] = 1,2,3,4,5,6,7,8,9;
matrix B[2][2] = 1,x,y,z;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(B);
↦ 1,x,
↦ y,z
print(dsum(A,B));
↦ 1,2,3,0,0,
↦ 4,5,6,0,0,
↦ 7,8,9,0,0,
↦ 0,0,0,1,x,
↦ 0,0,0,y,z
```

D.3.1.5 flatten

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: flatten(A); A matrix/smatrix

Return: ideal, generated by all entries from A resp. all columns of A appended

Example:

```
LIB "matrix.lib";
ring r = 0,(x,y,z),ds;
matrix A[2][3] = 1,2,x,y,z,7;
print(A);
↦ 1,2,x,
↦ y,z,7
flatten(A);
↦ _[1]=1
↦ _[2]=2
↦ _[3]=x
↦ _[4]=y
↦ _[5]=z
↦ _[6]=7
flatten(smatrix(A));
↦ 1,
↦ y,
↦ 2,
↦ z,
↦ x,
↦ 7
```

D.3.1.6 genericmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: genericmat(n,m[,id]); n,m=integers, id=ideal

Return: nxm matrix, with entries from id.

Note: if id has less than nxm elements, the matrix is filled with 0's, (default: id=maxideal(1)).
genericmat(n,m); creates the generic nxm matrix

Example:

```
LIB "matrix.lib";
ring R = 0,x(1..16),lp;
print(genericmat(3,3));      // the generic 3x3 matrix
↦ x(1),x(2),x(3),
↦ x(4),x(5),x(6),
↦ x(7),x(8),x(9)
ring R1 = 0,(a,b,c,d),dp;
matrix A = genericmat(3,4,maxideal(1)^3);
print(A);
↦ a3, a2b,a2c,a2d,
↦ ab2,abc,abd,ac2,
↦ acd,ad2,b3, b2c
int n,m = 3,2;
ideal i = ideal(randommat(1,n*m,maxideal(1),9));
print(genericmat(n,m,i));    // matrix of generic linear forms
↦ 4a-8b-2c-3d,-a+b-4c+5d,
↦ -8a-9b+c+7d,a-9b+9c+4d,
↦ 6a-5b+9c,   2a+8c+d
```

D.3.1.7 is_complex

Procedure from library matrix.lib (see Section D.3.1 [matrix_lib], page 967).

Usage: is_complex(c); c = list of size-compatible modules or matrices

Return: 1 if c[i]*c[i+1]=0 for all i, 0 if not, hence checking whether the list of matrices forms a complex.

Note: Ideals are treated internally as 1-line matrices.
If printlevel > 0, the position where c is not a complex is shown.

Example:

```
LIB "matrix.lib";
ring r  = 32003,(x,y,z),ds;
ideal i = x4+y5+z6,xyz,yx2+xz2+zy7;
list L  = nres(i,0);
is_complex(L);
↦ 1
L[4]    = matrix(i);
is_complex(L);
↦ 0
```

D.3.1.8 outer

Procedure from library matrix.lib (see Section D.3.1 [matrix_lib], page 967).

Usage: outer(A,B); A,B matrices

Return: matrix, outer (tensor) product of A and B

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),ds;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
matrix B[2][2]=x,y,0,z;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(B);
↦ x,y,
↦ 0,z
print(outer(A,B));
↦ x, y, 2x,2y,3x,3y,
↦ 0, z, 0, 2z,0, 3z,
↦ 4x,4y,5x,5y,6x,6y,
↦ 0, 4z,0, 5z,0, 6z,
↦ 7x,7y,8x,8y,9x,9y,
↦ 0, 7z,0, 8z,0, 9z
```

D.3.1.9 power

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: power(A,n); A a square-matrix of type intmat or matrix, n=integer>=0

Return: intmat resp. matrix, the n-th power of A

Note: for A=intmat and big n the result may be wrong because of int overflow

Example:

```
LIB "matrix.lib";
intmat A[3][3]=1,2,3,4,5,6,7,8,9;
print(power(A,3));"";
↦    468   576   684
↦   1062  1305  1548
↦   1656  2034  2412
↦ 
ring r=0,(x,y,z),dp;
matrix B[3][3]=0,x,y,z,0,0,y,z,0;
print(power(B,3));"";
↦ yz2,     xy2+x2z,y3+xyz,
↦ y2z+xz2,yz2,     0,
↦ y3+xyz, y2z+xz2,yz2
↦ 
```

D.3.1.10 skewmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: skewmat(n[,id]); n integer, id ideal

Return: skew-symmetric nxn matrix, with entries from id
(default: id=maxideal(1))
skewmat(n); creates the generic skew-symmetric matrix

Note: if id has less than n*(n-1)/2 elements, the matrix is
filled with 0's,

Example:

```
LIB "matrix.lib";
ring R=0,x(1..5),lp;
print(skewmat(4));    // the generic skew-symmetric matrix
↦ 0,    x(1), x(2),x(3),
↦ -x(1),0,    x(4),x(5),
↦ -x(2),-x(4),0,   0,
↦ -x(3),-x(5),0,   0
ring R1 = 0,(a,b,c),dp;
matrix A=skewmat(4,maxideal(1)^2);
print(A);
↦ 0,  a2, ab, ac,
↦ -a2,0,  b2, bc,
↦ -ab,-b2,0,  c2,
↦ -ac,-bc,-c2,0
int n=3;
ideal i = ideal(randommat(1,n*(n-1) div 2,maxideal(1),9));
print(skewmat(n,i));  // skew matrix of generic linear forms
↦ 0,       4a+b-8c, -a+6b+c,
↦ -4a-b+8c,0,       -8a+2b-9c,
↦ a-6b-c,  8a-2b+9c,0
kill R1;
```

D.3.1.11 submat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: submat(A,r,c); A=matrix, r,c=intvec

Return: matrix, submatrix of A with rows specified by intvec r and columns specified by intvec c.

Example:

```
LIB "matrix.lib";
ring R=32003,(x,y,z),lp;
matrix A[4][4]=x,y,z,0,1,2,3,4,5,6,7,8,9,x2,y2,z2;
print(A);
↦ x,y, z, 0,
↦ 1,2, 3, 4,
↦ 5,6, 7, 8,
↦ 9,x2,y2,z2
intvec v=1,3,4;
matrix B=submat(A,v,1..3);
print(B);
↦ x,y, z,
↦ 5,6, 7,
↦ 9,x2,y2
```

D.3.1.12 symmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: symmat(n[,id]); n integer, id ideal

Return: symmetric nxn matrix, with entries from id (default: id=maxideal(1))

Note: if id has less than n*(n+1)/2 elements, the matrix is filled with 0's, symmat(n); creates the generic symmetric matrix

Example:

```
LIB "matrix.lib";
ring R=0,x(1..10),lp;
print(symmat(4));    // the generic symmetric matrix
↦ x(1),x(2),x(3),x(4),
↦ x(2),x(5),x(6),x(7),
↦ x(3),x(6),x(8),x(9),
↦ x(4),x(7),x(9),x(10)
ring R1 = 0,(a,b,c),dp;
matrix A=symmat(4,maxideal(1)^3);
```

```
print(A);
↦ a3, a2b,a2c,ab2,
↦ a2b,abc,ac2,b3,
↦ a2c,ac2,b2c,bc2,
↦ ab2,b3, bc2,c3
int n=3;
ideal i = ideal(randommat(1,n*(n+1) div 2,maxideal(1),9));
print(symmat(n,i));  // symmetric matrix of generic linear forms
↦ 4a-8b-2c,-a+b-4c, -8a-9b+c,
↦ -a+b-4c, a-9b+9c, 6a-5b+9c,
↦ -8a-9b+c,6a-5b+9c,2a+8c
kill R1;
```

D.3.1.13 unitmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: unitmat(n); n integer >= 0

Return: nxn unit matrix

Note: needs a basering, diagonal entries are numbers (=1) in the basering

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
print(xyz*unitmat(4));
↦ xyz,0,  0,  0,
↦ 0,  xyz,0,  0,
↦ 0,  0,  xyz,0,
↦ 0,  0,  0,  xyz
print(unitmat(5));
↦ 1,0,0,0,0,
↦ 0,1,0,0,0,
↦ 0,0,1,0,0,
↦ 0,0,0,1,0,
↦ 0,0,0,0,1
```

D.3.1.14 gauss_col

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: gauss_col(A[,e]); A a matrix, e any type

Return: - a matrix B, if called with one argument; B is the complete column- reduced upper-triangular normal form of A if A is constant, (resp. as far as this is possible if A is a polynomial matrix; no division by polynomials).
- a list L of two matrices, if called with two arguments; L satisfies L[1] = A * L[2] with L[1] the column-reduced form of A and L[2] the transformation matrix.

Note: * The procedure just applies interred to A with ordering (C,dp). The transformation matrix is obtained by applying 'lift'. This should be faster than the procedure colred.
* It should only be used with exact coefficient field (there is no pivoting and rounding error treatment).
* Parameters are allowed. Hence, if the entries of A are parameters, B is the column-reduced form of A over the rational function field.

Example:

```
LIB "matrix.lib";
ring r=(0,a,b),(A,B,C),dp;
matrix m[8][6]=
0,    2*C, 0,    0,  0,   0,
0,    -4*C,a*A,  0,  0,   0,
b*B,  -A,  0,    0,  0,   0,
-A,   B,   0,    0,  0,   0,
-4*C, 0,   B,    2,  0,   0,
2*A,  B,   0,    0,  0,   0,
0,    3*B, 0,    0,  2b,  0,
0,    AB,  0,    2*A,A,   2a;"";
↦
list L=gauss_col(m,1);
print(L[1]);
↦ 0,0,2*C, 0,      0,0,
↦ A,0,-4*C,0,      0,0,
↦ 0,0,-A,  (b)/2*B,0,0,
↦ 0,0,B,   -1/2*A, 0,0,
↦ 0,1,0,   0,      0,0,
↦ 0,0,B,   A,      0,0,
↦ 0,0,0,   0,      1,0,
↦ 0,0,0,   0,      0,1
print(L[2]);
↦ 0,         0,        0,               1/2,      0,         0,
↦ 0,         0,        1,               0,        0,         0,
↦ 1/(a),     0,        0,               0,        0,         0,
↦ -1/(2a)*B, 1/2,      0,               C,        0,         0,
↦ 0,         0,        -3/(2b)*B,       0,        1/(2b),    0,
↦ 1/(2a2)*AB,-1/(2a)*A,(-2b+3)/(4ab)*AB,-1/(a)*AC,-1/(4ab)*A,1/(2a)
ring S=0,x,(c,dp);
matrix A[5][4] =
3, 1, 1, 1,
13, 8, 6,-7,
14,10, 6,-7,
7, 4, 3,-3,
2, 1, 0, 3;
print(gauss_col(A));
↦ 8/9,-5/9,-1/3,7/9,
↦ 1,  0,   0,   0,
↦ 0,  1,   0,   0,
↦ 0,  0,   1,   0,
↦ 0,  0,   0,   1
```

See also: Section D.3.1.23 [colred], page 980.

D.3.1.15 gauss_row

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: gauss_row(A [,e]); A matrix, e any type

Return: - a matrix B, if called with one argument; B is the complete row- reduced lower-triangular normal form of A if A is constant, (resp. as far as this is possible if A is a polynomial matrix; no division by polynomials).

- a list L of two matrices, if called with two arguments; L satisfies transpose(L[2])*A=transpose(L[1])
with L[1] the row-reduced form of A
and L[2] the transformation matrix.

Note: * This procedure just applies gauss_col to the transposed matrix. The transformation matrix is obtained by applying lift. This should be faster than the procedure rowred.
* It should only be used with exact coefficient field (there is no pivoting and rounding error treatment).
* Parameters are allowed. Hence, if the entries of A are parameters, B is the row-reduced form of A over the rational function field.

Example:

```
LIB "matrix.lib";
ring r=(0,a,b),(A,B,C),dp;
matrix m[6][8]=
0, 0,  b*B, -A,-4C,2A,0, 0,
2C,-4C,-A,B, 0,  B, 3B,AB,
0,a*A,  0, 0, B,  0, 0, 0,
0, 0,  0, 0, 2,  0, 0, 2A,
0, 0,  0, 0, 0,  0, 2b, A,
0, 0,  0, 0, 0,  0, 0, 2a;"";
↦
print(gauss_row(m));"";
↦ 0,  A,   0,       0,     0,0,0,0,
↦ 0,  0,   0,       0,     1,0,0,0,
↦ 2*C,-4*C,-A,      B,     0,B,0,0,
↦ 0,  0,   (b)/2*B,-1/2*A,0,A,0,0,
↦ 0,  0,   0,       0,     0,0,1,0,
↦ 0,  0,   0,       0,     0,0,0,1
↦
ring S=0,x,dp;
matrix A[4][5] =  3, 1,1,-1,2,
13, 8,6,-7,1,
14,10,6,-7,1,
7, 4,3,-3,3;
list L=gauss_row(A,1);
print(L[1]);
↦ 1/2,-7/3,-19/6,5/6,
↦ 1,  0,   0,    0,
↦ 0,  1,   0,    0,
↦ 0,  0,   1,    0,
↦ 0,  0,   0,    1
print(L[2]);
↦ 0,   -6,  -5,  1,
↦ -1/2,2/3, -1/6,-1/6,
↦ 1/2, -5/3,-5/6,1/6,
↦ 0,   13/3,11/3,-1/3
```

See also: Section D.3.1.22 [rowred], page 979.

D.3.1.16 addcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: addcol(A,c1,p,c2); A matrix, p poly, c1, c2 positive integers

Return: matrix, A being modified by adding p times column c1 to column c2

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(addcol(A,1,xy,2));
↦ 1,xy+2, 3,
↦ 4,4xy+5,6,
↦ 7,7xy+8,9
```

D.3.1.17 addrow

Procedure from library matrix.lib (see Section D.3.1 [matrix_lib], page 967).

Usage: addrow(A,r1,p,r2); A matrix, p poly, r1, r2 positive integers

Return: matrix, A being modified by adding p times row r1 to row r2

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(addrow(A,1,xy,3));
↦ 1,   2,    3,
↦ 4,   5,    6,
↦ xy+7,2xy+8,3xy+9
```

D.3.1.18 multcol

Procedure from library matrix.lib (see Section D.3.1 [matrix_lib], page 967).

Usage: multcol(A,c,p); A matrix, p poly, c positive integer

Return: matrix, A being modified by multiplying column c by p

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(multcol(A,2,xy));
↦ 1,2xy,3,
↦ 4,5xy,6,
↦ 7,8xy,9
```

D.3.1.19 multrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: multrow(A,r,p); A matrix, p poly, r positive integer

Return: matrix, A being modified by multiplying row r by p

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,2,3,4,5,6,7,8,9;
print(A);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(multrow(A,2,xy));
↦ 1,  2,  3,
↦ 4xy,5xy,6xy,
↦ 7,  8,  9
```

D.3.1.20 permcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: permcol(A,c1,c2); A matrix, c1,c2 positive integers

Return: matrix, A being modified by permuting columns c1 and c2

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,x,3,4,y,6,7,z,9;
print(A);
↦ 1,x,3,
↦ 4,y,6,
↦ 7,z,9
print(permcol(A,2,3));
↦ 1,3,x,
↦ 4,6,y,
↦ 7,9,z
```

D.3.1.21 permrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: permrow(A,r1,r2); A matrix, r1,r2 positive integers

Return: matrix, A being modified by permuting rows r1 and r2

Example:

```
LIB "matrix.lib";
ring r=32003,(x,y,z),lp;
matrix A[3][3]=1,2,3,x,y,z,7,8,9;
print(A);
↦ 1,2,3,
↦ x,y,z,
↦ 7,8,9
print(permrow(A,2,1));
↦ x,y,z,
↦ 1,2,3,
↦ 7,8,9
```

D.3.1.22 rowred

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: rowred(A[,e]); A matrix, e any type

Return: - a matrix B, being the row reduced form of A, if rowred is called with one argument. (as far as this is possible over the polynomial ring; no division by polynomials)
- a list L of two matrices, such that L[1] = L[2] * A with L[1] the row-reduced form of A and L[2] the transformation matrix (if rowred is called with two arguments).

Assume: The entries of A are in the base field. It is not verified whether this assumption holds.

Note: * This procedure is designed for teaching purposes mainly.
* The straight forward Gaussian algorithm is implemented in the library (no standard basis computation).
The transformation matrix is obtained by concatenating a unit matrix to A. proc gauss_row should be faster.
* It should only be used with exact coefficient field (there is no pivoting) over the polynomial ring (ordering lp or dp).
* Parameters are allowed. Hence, if the entries of A are parameters the computation takes place over the field of rational functions.

Example:

```
LIB "matrix.lib";
ring r=(0,a,b),(A,B,C),dp;
matrix m[6][8]=
0, 0,  b*B, -A,-4C,2A,0, 0,
2C,-4C,-A,B, 0,  B, 3B,AB,
0,a*A,  0, 0, B,  0, 0, 0,
0, 0,  0, 0, 2,  0, 0, 2A,
0, 0,  0, 0, 0,  0, 2b, A,
0, 0,  0, 0, 0,  0, 0, 2a;"";
↦
print(rowred(m));"";
↦ 0,  0,     0,     0, 1,0,  0,0,
↦ 0,  0,     0,     0, 0,0,  1,0,
↦ 0,  0,     0,     0, 0,0,  0,1,
↦ 0,  0,     (b)*B,-A,0,2*A,0,0,
↦ 2*C,-4*C, -A,   B, 0,B,  0,0,
↦ 0,  (a)*A,0,    0, 0,0,  0,0
↦
list L=rowred(m,1);
print(L[1]);
↦ 0,  0,     0,     0, 1,0,  0,0,
↦ 0,  0,     0,     0, 0,0,  1,0,
↦ 0,  0,     0,     0, 0,0,  0,1,
↦ 0,  0,     (b)*B,-A,0,2*A,0,0,
↦ 2*C,-4*C, -A,   B, 0,B,  0,0,
↦ 0,  (a)*A,0,    0, 0,0,  0,0
print(L[2]);
↦ 0,0,0,1/2,   0,         -1/(2a)*A,
↦ 0,0,0,0,     1/(2b),    -1/(4ab)*A,
↦ 0,0,0,0,     0,         1/(2a),
↦ 1,0,0,2*C,   0,         -2/(a)*AC,
↦ 0,1,0,0,     -3/(2b)*B,(-2b+3)/(4ab)*AB,
↦ 0,0,1,-1/2*B,0,         1/(2a)*AB
```

See also: Section D.3.1.15 [gauss_row], page 975.

D.3.1.23 colred

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: colred(A[,e]); A matrix, e any type

Return: - a matrix B, being the column reduced form of A, if colred is called with one argument. (as far as this is possible over the polynomial ring; no division by polynomials)
- a list L of two matrices, such that L[1] = A * L[2] with L[1] the column-reduced form of A and L[2] the transformation matrix (if colred is called with two arguments).

Assume: The entries of A are in the base field. It is not verified whether this assumption holds.

Note: * This procedure is designed for teaching purposes mainly.
* It applies rowred to the transposed matrix. proc gauss_col should be faster.
* It should only be used with exact coefficient field (there is no pivoting) over the polynomial ring (ordering lp or dp).
* Parameters are allowed. Hence, if the entries of A are parameters the computation takes place over the field of rational functions.

Example:

```
LIB "matrix.lib";
ring r=(0,a,b),(A,B,C),dp;
matrix m[8][6]=
0,    2*C, 0,    0,  0,   0,
0,    -4*C,a*A,  0,  0,   0,
b*B,  -A,  0,    0,  0,   0,
-A,   B,   0,    0,  0,   0,
-4*C, 0,   B,    2,  0,   0,
2*A,  B,   0,    0,  0,   0,
0,    3*B, 0,    0,  2b,  0,
0,    AB,  0,    2*A,A,   2a;"";
↦
print(colred(m));"";
↦ 0,0,0,0,    2*C, 0,
↦ 0,0,0,0,    -4*C,(a)*A,
↦ 0,0,0,(b)*B,-A,  0,
↦ 0,0,0,-A,   B,   0,
↦ 1,0,0,0,    0,   0,
↦ 0,0,0,2*A,  B,   0,
↦ 0,1,0,0,    0,   0,
↦ 0,0,1,0,    0,   0
↦
list L=colred(m,1);
print(L[1]);
```

```
↦ 0,0,0,0,    2*C, 0,
↦ 0,0,0,0,    -4*C,(a)*A,
↦ 0,0,0,(b)*B,-A,  0,
↦ 0,0,0,-A,   B,   0,
↦ 1,0,0,0,    0,   0,
↦ 0,0,0,2*A,  B,   0,
↦ 0,1,0,0,    0,   0,
↦ 0,0,1,0,    0,   0
print(L[2]);
↦ 0,        0,         0,     1,        0,                0,
↦ 0,        0,         0,     0,        1,                0,
↦ 0,        0,         0,     0,        0,                1,
↦ 1/2,      0,         0,     2*C,      0,                -1/2*B,
↦ 0,        1/(2b),    0,     0,        -3/(2b)*B,        0,
↦ -1/(2a)*A,-1/(4ab)*A,1/(2a),-2/(a)*AC,(-2b+3)/(4ab)*AB,1/(2a)*AB
```

See also: Section D.3.1.14 [gauss_col], page 974.

D.3.1.24 linear_relations

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: linear_relations(M);
M: a module

Assume: All non-zero entries of M are homogeneous polynomials of the same positive degree.
The base field must be an exact field (not real or complex).
It is not checked whether these assumptions hold.

Return: a maximal module R such that M*R is formed by zero vectors.

Example:

```
LIB "matrix.lib";
ring r = (3,w), (a,b,c,d),dp;
minpoly = w2-w-1;
module M = [a2,b2],[wab,w2c2+2b2],[(w-2)*a2+wab,wb2+w2c2];
module REL = linear_relations(M);
pmat(REL);
↦ (-w-1),
↦ -1,
↦ 1
pmat(matrix(M)*matrix(REL));
↦ 0,
↦ 0
```

D.3.1.25 rm_unitrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: rm_unitrow(A); A matrix (being col-reduced)

Return: matrix, obtained from A by deleting unit rows (having just one 1 and else 0 as entries) and associated columns

Example:

```
LIB "matrix.lib";
ring r=0,(A,B,C),dp;
matrix m[8][6]=
0,0,  0,   0, 2C, 0,
0,0,  0,   0, -4C,A,
A,-C2,0,   B, -A, 0,
0,0,  1/2B,-A,B,  0,
1,0,  0,   0, 0,  0,
0,0,  0,   2A,B,  0,
0,1,  0,   0, 0,  0,
0,0,  1,   0, 0,  0;
print(rm_unitrow(m));
↦ 0, 2C, 0,
↦ 0, -4C,A,
↦ B, -A, 0,
↦ -A,B,  0,
↦ 2A,B,  0
```

D.3.1.26 rm_unitcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: rm_unitcol(A); A matrix (being row-reduced)

Return: matrix, obtained from A by deleting unit columns (having just one 1 and else 0 as entries) and associated rows

Example:

```
LIB "matrix.lib";
ring r=0,(A,B,C),dp;
matrix m[6][8]=
0,  0,    A,   0, 1,0,  0,0,
0,  0,  -C2,   0, 0,0,  1,0,
0,  0,    0,1/2B, 0,0,  0,1,
0,  0,    B,  -A, 0,2A, 0,0,
2C,-4C,  -A,   B, 0,B,  0,0,
0,  A,    0,   0, 0,0,  0,0;
print(rm_unitcol(m));
↦ 0, 0,  B, -A,2A,
↦ 2C,-4C,-A,B, B,
↦ 0, A,  0, 0, 0
```

D.3.1.27 headStand

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: headStand(M); M matrix

Return: matrix B such that B[i][j]=M[n-i+1,m-j+1], n=nrows(M), m=ncols(M)

Example:

```
LIB "matrix.lib";
ring r=0,(A,B,C),dp;
matrix M[2][3]=
0,A,  B,
A2, B2, C;
```

```
print(M);
↦ 0, A, B,
↦ A2,B2,C
print(headStand(M));
↦ C,B2,A2,
↦ B,A, 0
```

D.3.1.28 symmetricBasis

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Return: ring, poynomial ring containing the ideal "symBasis", being a basis of the k-th symmetric power of an n-dim vector space.

Note: The output polynomial ring has characteristics 0 and n variables named "S(i)", where the base variable name S is either given by the optional string argument(which must not contain brackets) or equal to "e" by default.

Example:

```
LIB "matrix.lib";
// basis of the 3-rd symmetricPower of a 4-dim vector space:
def R = symmetricBasis(4, 3, "@e"); setring R;
R;  // container ring:
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    @e(1) @e(2) @e(3) @e(4)
↦ //        block   2 : ordering C
symBasis; // symmetric basis:
↦ symBasis[1]=@e(4)^3
↦ symBasis[2]=@e(3)*@e(4)^2
↦ symBasis[3]=@e(3)^2*@e(4)
↦ symBasis[4]=@e(3)^3
↦ symBasis[5]=@e(2)*@e(4)^2
↦ symBasis[6]=@e(2)*@e(3)*@e(4)
↦ symBasis[7]=@e(2)*@e(3)^2
↦ symBasis[8]=@e(2)^2*@e(4)
↦ symBasis[9]=@e(2)^2*@e(3)
↦ symBasis[10]=@e(2)^3
↦ symBasis[11]=@e(1)*@e(4)^2
↦ symBasis[12]=@e(1)*@e(3)*@e(4)
↦ symBasis[13]=@e(1)*@e(3)^2
↦ symBasis[14]=@e(1)*@e(2)*@e(4)
↦ symBasis[15]=@e(1)*@e(2)*@e(3)
↦ symBasis[16]=@e(1)*@e(2)^2
↦ symBasis[17]=@e(1)^2*@e(4)
↦ symBasis[18]=@e(1)^2*@e(3)
↦ symBasis[19]=@e(1)^2*@e(2)
↦ symBasis[20]=@e(1)^3
```

See also: Section D.3.1.29 [exteriorBasis], page 983.

D.3.1.29 exteriorBasis

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Return: qring, an exterior algebra containing the ideal "extBasis", being a basis of the k-th exterior power of an n-dim vector space.

Note: The output polynomial ring has characteristics 0 and n variables named "S(i)", where the base variable name S is either given by the optional string argument(which must not contain brackets) or equal to "e" by default.

Example:

```
LIB "matrix.lib";
// basis of the 3-rd symmetricPower of a 4-dim vector space:
def r = exteriorBasis(4, 3, "@e"); setring r;
r; // container ring:
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    @e(1) @e(2) @e(3) @e(4)
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    @e(2)@e(1)=-@e(1)*@e(2)
↦ //    @e(3)@e(1)=-@e(1)*@e(3)
↦ //    @e(4)@e(1)=-@e(1)*@e(4)
↦ //    @e(3)@e(2)=-@e(2)*@e(3)
↦ //    @e(4)@e(2)=-@e(2)*@e(4)
↦ //    @e(4)@e(3)=-@e(3)*@e(4)
↦ // quotient ring from ideal
↦ _[1]=@e(4)^2
↦ _[2]=@e(3)^2
↦ _[3]=@e(2)^2
↦ _[4]=@e(1)^2
extBasis; // exterior basis:
↦ extBasis[1]=@e(2)*@e(3)*@e(4)
↦ extBasis[2]=@e(1)*@e(3)*@e(4)
↦ extBasis[3]=@e(1)*@e(2)*@e(4)
↦ extBasis[4]=@e(1)*@e(2)*@e(3)
```

See also: Section D.3.1.28 [symmetricBasis], page 983.

D.3.1.30 symmetricPower

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: symmetricPower(A, k); A module, k int

Return: module: the k-th symmetric power of A

Note: the chosen bases and most of intermediate data will be shown if printlevel is big enough

Example:

```
LIB "matrix.lib";
ring r = (0),(a, b, c, d), dp; r;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    a b c d
↦ //        block   2 : ordering C
module B = a*gen(1) + c* gen(2), b * gen(1) + d * gen(2); print(B);
```

```
↦ a,b,
↦ c,d
// symmetric power over a commutative K-algebra:
print(symmetricPower(B, 2));
↦ d2, cd,   c2,
↦ 2bd,bc+ad,2ac,
↦ b2, ab,   a2
print(symmetricPower(B, 3));
↦ d3,  cd2,     c2d,     c3,
↦ 3bd2,2bcd+ad2,bc2+2acd,3ac2,
↦ 3b2d,b2c+2abd,2abc+a2d,3a2c,
↦ b3,  ab2,     a2b,     a3
// symmetric power over an exterior algebra:
def g = superCommutative(); setring g; g;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    a b c d
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=-ab
↦ //    ca=-ac
↦ //    da=-ad
↦ //    cb=-bc
↦ //    db=-bd
↦ //    dc=-cd
↦ // quotient ring from ideal
↦ _[1]=d2
↦ _[2]=c2
↦ _[3]=b2
↦ _[4]=a2
module B = a*gen(1) + c* gen(2), b * gen(1) + d * gen(2); print(B);
↦ a,b,
↦ c,d
print(symmetricPower(B, 2)); // much smaller!
↦ 0,cd,    0,
↦ 0,-bc+ad,0,
↦ 0,ab,    0
print(symmetricPower(B, 3)); // zero! (over an exterior algebra!)
↦ 0,0,0,0
```

See also: Section D.3.1.31 [exteriorPower], page 985.

D.3.1.31 exteriorPower

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 967).

Usage: exteriorPower(A, k); A module, k int

Return: module: the k-th exterior power of A

Note: the chosen bases and most of intermediate data will be shown if printlevel is big enough. Last rows will be invisible if zero.

Example:

```
LIB "matrix.lib";
ring r = (0),(a, b, c, d, e, f), dp;
r; "base ring:";
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    a b c d e f
↦ //        block   2 : ordering C
↦ base ring:
module B = a*gen(1) + c*gen(2) + e*gen(3),
b*gen(1) + d*gen(2) + f*gen(3),
e*gen(1) + f*gen(3);
print(B);
↦ a,b,e,
↦ c,d,0,
↦ e,f,f
print(exteriorPower(B, 2));
↦ df,   cf,    -de+cf,
↦ bf-ef,-e2+af,-be+af,
↦ -de,  -ce,   -bc+ad
print(exteriorPower(B, 3));
↦ -de2-bcf+adf+cef
def g = superCommutative(); setring g; g;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    a b c d e f
↦ //        block   2 : ordering C
↦ // noncommutative relations:
↦ //    ba=-ab
↦ //    ca=-ac
↦ //    da=-ad
↦ //    ea=-ae
↦ //    fa=-af
↦ //    cb=-bc
↦ //    db=-bd
↦ //    eb=-be
↦ //    fb=-bf
↦ //    dc=-cd
↦ //    ec=-ce
↦ //    fc=-cf
↦ //    ed=-de
↦ //    fd=-df
↦ //    fe=-ef
↦ // quotient ring from ideal
↦ _[1]=f2
↦ _[2]=e2
↦ _[3]=d2
↦ _[4]=c2
↦ _[5]=b2
↦ _[6]=a2
module A = a*gen(1), b * gen(1), c*gen(2), d * gen(2);
print(A);
```

```
↦ a,b,0,0,
↦ 0,0,c,d
print(exteriorPower(A, 2));
↦ 0,bd,bc,ad,ac,0
module B = a*gen(1) + c*gen(2) + e*gen(3),
b*gen(1) + d*gen(2) + f*gen(3),
e*gen(1) + f*gen(3);
print(B);
↦ a,b,e,
↦ c,d,0,
↦ e,f,f
print(exteriorPower(B, 2));
↦ df,   cf, de+cf,
↦ bf+ef,af, be+af,
↦ -de,  -ce,bc+ad
print(exteriorPower(B, 3));
↦ bcf+adf-cef
```

See also: Section D.3.1.30 [symmetricPower], page 984.

D.3.2 linalg_lib

Library: linalg.lib

Purpose: Algorithmic Linear Algebra

Authors: Ivor Saynisch (ivs@math.tu-cottbus.de)
Mathias Schulze (mschulze@mathematik.uni-kl.de)

Procedures:

D.3.2.1 inverse

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: inverse(A [,opt]); A a square matrix, opt integer

Return:

a matrix:
- the inverse matrix of A, if A is invertible;
- the 1x1 0-matrix if A is not invertible (in the polynomial ring!).
There are the following options:
- opt=0 or not given: heuristically best option from below
- opt=1 : apply std to (transpose(E,A)), ordering (C,dp).
- opt=2 : apply interred (transpose(E,A)), ordering (C,dp).
- opt=3 : apply lift(A,E), ordering (C,dp).

Note: parameters and minpoly are allowed; opt=2 is only correct for matrices with entries in a field

Example:

```
LIB "linalg.lib";
ring r=0,(x,y,z),lp;
matrix A[3][3]=
1,4,3,
1,5,7,
0,4,17;
print(inverse(A));"";
↦ 57, -56,13,
↦ -17,17, -4,
↦ 4,  -4, 1
↦
matrix B[3][3]=
y+1,  x+y,    y,
z,    z+1,    z,
y+z+2,x+y+z+2,y+z+1;
print(inverse(B));
↦ -xz+y+1, -xz-x+y,   xz-y,
↦ z,       z+1,       -z,
↦ xz-y-z-2,xz+x-y-z-2,-xz+y+z+1
print(B*inverse(B));
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
```

See also: Section D.3.2.2 [inverse_B], page 988; Section D.3.2.3 [inverse_L], page 989.

D.3.2.2 inverse_B

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: inverse_B(A); A = square matrix

Return: list Inv with
- Inv[1] = matrix I and
- Inv[2] = poly p
such that I*A = unitmat(n)*p;

Note: p=1 if 1/det(A) is computable and p=det(A) if not;
the computation uses busadj.

Example:

```
LIB "linalg.lib";
ring r=0,(x,y),lp;
matrix A[3][3]=x,y,1,1,x2,y,x,6,0;
print(A);
↦ x,y, 1,
↦ 1,x2,y,
↦ x,6, 0
list Inv=inverse_B(A);
print(Inv[1]);
↦ 6y,  -6,    x2-y2,
↦ -xy, x,     xy-1,
↦ x3-6,-xy+6x,-x3+y
print(Inv[2]);
↦ x3-xy2+6xy-6
print(Inv[1]*A);
↦ x3-xy2+6xy-6,0,           0,
↦ 0,           x3-xy2+6xy-6,0,
↦ 0,           0,           x3-xy2+6xy-6
```

See also: Section D.3.2.1 [inverse], page 987; Section D.3.2.3 [inverse_L], page 989.

D.3.2.3 inverse_L

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: inverse_L(A); A = square matrix

Return: list Inv representing a left inverse of A, i.e
- Inv[1] = matrix I and
- Inv[2] = poly p
such that I*A = unitmat(n)*p;

Note: p=1 if 1/det(A) is computable and p=det(A) if not;
the computation computes first det(A) and then uses lift

Example:

```
LIB "linalg.lib";
ring r=0,(x,y),lp;
matrix A[3][3]=x,y,1,1,x2,y,x,6,0;
print(A);
↦ x,y, 1,
↦ 1,x2,y,
↦ x,6, 0
list Inv=inverse_L(A);
print(Inv[1]);
↦ -6y,  6,    -x2+y2,
↦ xy,   -x,   -xy+1,
↦ -x3+6,xy-6x,x3-y
print(Inv[2]);
↦ -x3+xy2-6xy+6
print(Inv[1]*A);
↦ -x3+xy2-6xy+6,0,            0,
↦ 0,            -x3+xy2-6xy+6,0,
↦ 0,            0,            -x3+xy2-6xy+6
```

See also: Section D.3.2.1 [inverse], page 987; Section D.3.2.2 [inverse_B], page 988.

D.3.2.4 sym_gauss

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: sym_gauss(A); A = symmetric matrix

Return: matrix, diagonalisation of A with symmetric gauss algorithm

Example:

```
LIB "linalg.lib";
ring r=0,(x),lp;
matrix A[2][2]=1,4,4,15;
print(A);
↦ 1,4,
↦ 4,15
print(sym_gauss(A));
↦ 1,0,
↦ 0,-1
```

D.3.2.5 orthogonalize

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: orthogonalize(A); A = matrix of constants

Return: matrix, orthogonal basis of the column space of A

Example:

```
LIB "linalg.lib";
ring r=0,(x),lp;
matrix A[4][4]=5,6,12,4,7,3,2,6,12,1,1,2,6,4,2,10;
print(A);
↦ 5, 6,12,4,
↦ 7, 3,2, 6,
↦ 12,1,1, 2,
↦ 6, 4,2, 10
print(orthogonalize(A));
↦ 1,0, 0, 0,
↦ 0,23,0, 0,
↦ 0,0, 21,0,
↦ 0,0, 0, 6
```

D.3.2.6 diag_test

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: diag_test(A); A = const square matrix

Return: int, 1 if A is diagonalizable,
0 if not
-1 if no statement is possible, since A does not split.

Note: The test works only for split matrices, i.e if eigenvalues of A are in the ground field. Does not work with parameters (uses factorize,gcd).

Example:

```
LIB "linalg.lib";
ring r=0,(x),dp;
matrix A[4][4]=6,0,0,0,0,0,6,0,0,6,0,0,0,0,0,6;
print(A);
↦ 6,0,0,0,
↦ 0,0,6,0,
↦ 0,6,0,0,
↦ 0,0,0,6
diag_test(A);
↦ 1
```

D.3.2.7 busadj

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: busadj(A); A = square matrix (of size nxn)

Return: list L:

L[1] contains the (n+1) coefficients of the characteristic
polynomial X of A, i.e.
X = L[1][1]+..+L[1][k]*t^(k-1)+..+(L[1][n+1])*t^n
L[2] contains the n (nxn)-matrices Hk which are the coefficients of
the busadjoint bA = adjoint(E*t-A) of A, i.e.
bA = (Hn-1)*t^(n-1)+...+Hk*t^k+...+H0, (Hk=L[2][k+1])

Example:

```
LIB "linalg.lib";
ring r = 0,(t,x),lp;
matrix A[2][2] = 1,x2,x,x2+3x;
print(A);
↦ 1,x2,
↦ x,x2+3x
list L = busadj(A);
poly X = L[1][1]+L[1][2]*t+L[1][3]*t2; X;
↦ t2-tx2-3tx-t-x3+x2+3x
matrix bA[2][2] = L[2][1]+L[2][2]*t;
print(bA);               //the busadjoint of A;
↦ t-x2-3x,x2,
↦ x,      t-1
print(bA*(t*unitmat(2)-A));
↦ t2-tx2-3tx-t-x3+x2+3x,0,
↦ 0,                    t2-tx2-3tx-t-x3+x2+3x
```

D.3.2.8 charpoly

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: charpoly(A[,v]); A square matrix, v string, name of a variable

Return: poly, the characteristic polynomial det(E*v-A)
(default: v=name of last variable)

Note: A must be independent of the variable v. The computation uses det. If printlevel>0, det(E*v-A) is displayed recursively.

Example:

```
LIB "linalg.lib";
ring r=0,(x,t),dp;
matrix A[3][3]=1,x2,x,x2,6,4,x,4,1;
print(A);
↦ 1, x2,x,
↦ x2,6, 4,
↦ x, 4, 1
charpoly(A,"t");
↦ -x4t+x4-8x3-x2t+t3+6x2-8t2-3t+10
```

D.3.2.9 adjoint

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: adjoint(A); A = square matrix

Return: adjoint matrix of A, i.e. Adj*A=det(A)*E

Note: computation uses busadj(A)

Example:

```
LIB "linalg.lib";
ring r=0,(t,x),lp;
matrix A[2][2]=1,x2,x,x2+3x;
print(A);
↦ 1,x2,
↦ x,x2+3x
matrix Adj[2][2]=adjoint(A);
print(Adj);                    //Adj*A=det(A)*E
↦ x2+3x,-x2,
↦ -x,   1
print(Adj*A);
↦ -x3+x2+3x,0,
↦ 0,        -x3+x2+3x
```

D.3.2.10 det_B

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: det_B(A); A any matrix

Return: returns the determinant of A

Note: the computation uses the busadj algorithm

Example:

```
LIB "linalg.lib";
ring r=0,(x),dp;
matrix A[10][10]=random(2,10,10)+unitmat(10)*x;
print(A);
↦ x+2,-1, 2, 0,  -1, 1, 1,  2,  -1, 1,
↦ 2,  x-1,-1,2,  -1, 1, 1,  2,  1,  0,
↦ -2, 2,  x, 1,  2,  1, -1, -2, -2, 2,
↦ 0,  -1, -1,x+2,0,  -1,1,  0,  -1, 0,
↦ 0,  1,  -2,0,  x+2,1, 1,  -2, 2,  1,
↦ 2,  -2, 0, -2, -1, x, -2, 1,  -2, -2,
↦ -1, 2,  2, 1,  1,  -2,x+1,-1, -2, -2,
↦ 1,  1,  -1,2,  -2, -1,2,  x+1,0,  1,
↦ -2, 0,  1, 2,  1,  -2,-2, 1,  x+1,0,
↦ 1,  2,  1, 2,  0,  1, -1, 1,  -1, x-1
det_B(A);
↦ x10+7x9+7x8-39x7-290x6-1777x5-3646x4+3725x3-5511x2-34811x-13241
```

D.3.2.11 gaussred

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: gaussred(A); A any constant matrix

Return: list Z: Z[1]=P , Z[2]=U , Z[3]=S , Z[4]=rank(A)
gives a row reduced matrix S, a permutation matrix P and a normalized lower triangular matrix U, with P*A=U*S

Note: This procedure is designed for teaching purposes mainly. The straight forward implementation in the interpreted library is not very efficient (no standard basis computation).

Example:

```
LIB "linalg.lib";
ring r=0,(x),dp;
matrix A[5][4]=1,3,-1,4,2,5,-1,3,1,3,-1,4,0,4,-3,1,-3,1,-5,-2;
print(A);
↦ 1, 3,-1,4,
↦ 2, 5,-1,3,
↦ 1, 3,-1,4,
↦ 0, 4,-3,1,
↦ -3,1,-5,-2
list Z=gaussred(A);   //construct P,U,S s.t. P*A=U*S
print(Z[1]);          //P
↦ 1,0,0,0,0,
↦ 0,1,0,0,0,
↦ 0,0,0,0,1,
↦ 0,0,0,1,0,
↦ 0,0,1,0,0
print(Z[2]);          //U
↦ 1, 0,  0,  0,0,
↦ 2, 1,  0,  0,0,
↦ -3,-10,1,  0,0,
↦ 0, -4, 1/2,1,0,
↦ 1, 0,  0,  0,1
print(Z[3]);          //S
↦ 1,3, -1,4,
↦ 0,-1,1, -5,
↦ 0,0, 2, -40,
↦ 0,0, 0, 1,
↦ 0,0, 0, 0
print(Z[4]);          //rank
↦ 4
print(Z[1]*A);        //P*A
↦ 1, 3,-1,4,
↦ 2, 5,-1,3,
↦ -3,1,-5,-2,
↦ 0, 4,-3,1,
↦ 1, 3,-1,4
print(Z[2]*Z[3]);     //U*S
↦ 1, 3,-1,4,
↦ 2, 5,-1,3,
↦ -3,1,-5,-2,
↦ 0, 4,-3,1,
↦ 1, 3,-1,4
```

D.3.2.12 gaussred_pivot

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: gaussred_pivot(A); A any constant matrix

Return: list Z: Z[1]=P , Z[2]=U , Z[3]=S , Z[4]=rank(A)
gives a row reduced matrix S, a permutation matrix P and a normalized lower triangular matrix U, with P*A=U*S

Note: with row pivoting

Example:

```
LIB "linalg.lib";
ring r=0,(x),dp;
matrix A[5][4] = 1, 3,-1,4,
2, 5,-1,3,
1, 3,-1,4,
0, 4,-3,1,
-3,1,-5,-2;
list Z=gaussred_pivot(A);  //construct P,U,S s.t. P*A=U*S
print(Z[1]);               //P
↦ 0,0,0,0,1,
↦ 0,1,0,0,0,
↦ 0,0,1,0,0,
↦ 0,0,0,1,0,
↦ 1,0,0,0,0
print(Z[2]);               //U
↦ 1,   0,    0,   0,0,
↦ -2/3,1,    0,   0,0,
↦ -1/3,10/17,1,   0,0,
↦ 0,   12/17,-1/2,1,0,
↦ -1/3,10/17,1,   0,1
print(Z[3]);               //S
↦ -3,1,   -5,   -2,
↦ 0, 17/3,-13/3,5/3,
↦ 0, 0,   -2/17,40/17,
↦ 0, 0,   0,    1,
↦ 0, 0,   0,    0
print(Z[4]);               //rank
↦ 4
print(Z[1]*A);             //P*A
↦ -3,1,-5,-2,
↦ 2, 5,-1,3,
↦ 1, 3,-1,4,
↦ 0, 4,-3,1,
↦ 1, 3,-1,4
print(Z[2]*Z[3]);          //U*S
↦ -3,1,-5,-2,
↦ 2, 5,-1,3,
↦ 1, 3,-1,4,
↦ 0, 4,-3,1,
↦ 1, 3,-1,4
```

D.3.2.13 gauss_nf

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: gauss_nf(A); A any constant matrix

Return: matrix; gauss normal form of A (uses gaussred)

Example:

```
LIB "linalg.lib";
ring r = 0,(x),dp;
matrix A[4][4] = 1,4,4,7,2,5,5,4,4,1,1,3,0,2,2,7;
```

```
print(gauss_nf(A));
↦ 1,4, 4, 7,
↦ 0,-3,-3,-10,
↦ 0,0, 0, 25,
↦ 0,0, 0, 0
```

D.3.2.14 mat_rk

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: mat_rk(A); A any constant matrix

Return: int, rank of A

Example:

```
LIB "linalg.lib";
ring r = 0,(x),dp;
matrix A[4][4] = 1,4,4,7,2,5,5,4,4,1,1,3,0,2,2,7;
mat_rk(A);
↦ 3
```

D.3.2.15 U_D_O

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: U_D_O(A); constant invertible matrix A

Return: list Z: Z[1]=P , Z[2]=U , Z[3]=D , Z[4]=O
gives a permutation matrix P,
a normalized lower triangular matrix U ,
a diagonal matrix D, and
a normalized upper triangular matrix O
with P*A=U*D*O

Note: Z[1]=-1 means that A is not regular (proc uses gaussred)

Example:

```
LIB "linalg.lib";
ring r = 0,(x),dp;
matrix A[5][5] = 10, 4,  0, -9,  8,
-3, 6, -6, -4,  9,
0, 3, -1, -9, -8,
-4,-2, -6, -10,10,
-9, 5, -1, -6,  5;
list Z = U_D_O(A);              //construct P,U,D,O s.t. P*A=U*D*O
print(Z[1]);                    //P
↦ 1,0,0,0,0,
↦ 0,1,0,0,0,
↦ 0,0,1,0,0,
↦ 0,0,0,1,0,
↦ 0,0,0,0,1
print(Z[2]);                    //U
↦ 1,     0,     0,     0,          0,
↦ -3/10,1,     0,     0,          0,
↦ 0,     5/12, 1,     0,          0,
↦ -2/5, -1/18,-38/9,1,          0,
↦ -9/10,43/36,37/9, -1049/2170,1
print(Z[3]);                    //D
↦ 10,0,   0,  0,        0,
↦ 0, 36/5,0,  0,        0,
↦ 0, 0,   3/2,0,        0,
↦ 0, 0,   0,  -1085/27,0,
↦ 0, 0,   0,  0,        6871/217
print(Z[4]);                    //O
↦ 1,2/5,0,   -9/10,  4/5,
↦ 0,1,  -5/6,-67/72, 19/12,
↦ 0,0,  1,   -149/36,-17/2,
↦ 0,0,  0,   1,      216/217,
↦ 0,0,  0,   0,      1
print(Z[1]*A);                  //P*A
↦ 10,4, 0, -9, 8,
↦ -3,6, -6,-4, 9,
↦ 0, 3, -1,-9, -8,
↦ -4,-2,-6,-10,10,
↦ -9,5, -1,-6, 5
print(Z[2]*Z[3]*Z[4]);          //U*D*O
↦ 10,4, 0, -9, 8,
↦ -3,6, -6,-4, 9,
↦ 0, 3, -1,-9, -8,
↦ -4,-2,-6,-10,10,
↦ -9,5, -1,-6, 5
```

D.3.2.16 pos_def

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: pos_def(A); A = constant, symmetric square matrix

Return: int:
1 if A is positive definit ,
0 if not,
-1 if unknown

Example:

```
LIB "linalg.lib";
ring r = 0,(x),dp;
matrix A[5][5] = 20,  4,  0, -9,   8,
4, 12, -6, -4,   9,
0, -6, -2, -9,  -8,
-9, -4, -9, -20, 10,
8,  9, -8,  10, 10;
pos_def(A);
↦ 0
matrix B[3][3] =  3,  2,  0,
2, 12,  4,
0,  4,  2;
pos_def(B);
↦ 1
```

D.3.2.17 hessenberg

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: hessenberg(M); matrix M

Assume: M constant square matrix

Return: matrix H; Hessenberg form of M

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
print(hessenberg(M));
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
```

See also: Section D.3.2.17 [hessenberg], page 997; Section 5.1.155 [system], page 275.

D.3.2.18 eigenvals

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: eigenvals(M); matrix M

Assume: eigenvalues of M in basefield

Return:

list l;
  ideal l[1];
    number l[1][i]; i-th eigenvalue of M
  intvec l[2];
    int l[2][i]; multiplicity of i-th eigenvalue of M

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
eigenvals(M);
↦ [1]:
↦    _[1]=2
↦    _[2]=3
↦ [2]:
↦    1,2
```

See also: Section D.3.2.18 [eigenvals], page 997; Section 5.1.155 [system], page 275.

D.3.2.19 minipoly

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: minipoly(M); matrix M

Assume: eigenvalues of M in basefield

Return:

list l; minimal polynomial of M
  ideal l[1];
    number l[1][i]; i-th root of minimal polynomial of M
  intvec l[2];
    int l[2][i]; multiplicity of i-th root of minimal polynomial of M

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
minipoly(M);
↦ [1]:
↦    _[1]=2
↦    _[2]=3
↦ [2]:
↦    1,2
```

D.3.2.20 spnf

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: spnf(list(a[,m])); ideal a, intvec m

Assume: ncols(a)==size(m)

Return: list l:
l[1] an ideal, the generators of a; sorted and with multiple entries displayed only once
l[2] an intvec, l[2][i] provides the multiplicity of l[1][i]

Example:

```
LIB "linalg.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-3/10,-1/10,-1/10,0,1/10,1/10,3/10,3/10,1/2));
spprint(spnf(sp));
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
```

D.3.2.21 spprint

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: spprint(sp); list sp (helper routine for spnf)

Return: string s; spectrum sp

Example:

```
LIB "linalg.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
```

See also: Section D.6.13 [gmssing_lib], page 1702; Section D.3.2.20 [spnf], page 998.

D.3.2.22 jordan

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: jordan(M); matrix M

Assume: eigenvalues of M in basefield

Return:

list l; Jordan data of M
  ideal l[1];
    number l[1][i]; eigenvalue of i-th Jordan block of M
  intvec l[2];
    int l[2][i]; size of i-th Jordan block of M
  intvec l[3];
    int l[3][i]; multiplicity of i-th Jordan block of M

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
jordan(M);
↦ [1]:
↦    _[1]=2
↦    _[2]=3
↦ [2]:
↦    1,2
↦ [3]:
↦    1,1
```

D.3.2.23 jordanbasis

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: jordanbasis(M); matrix M

Assume: eigenvalues of M in basefield

Return:

list l:
  module l[1]; inverse(l[1])*M*l[1] in Jordan normal form
  intvec l[2];
    int l[2][i]; weight filtration index of l[1][i]

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
list l=jordanbasis(M);
print(l[1]);
↦ -2,0,3,
↦ 1, 1,0,
↦ 0, 1,0
print(l[2]);
↦ 0,
↦ 1,
↦ -1
print(inverse(l[1])*M*l[1]);
↦ 2,0,0,
↦ 0,3,0,
↦ 0,1,3
```

D.3.2.24 jordanmatrix

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: jordanmatrix(list(e,s,m)); ideal e, intvec s, intvec m

Assume: ncols(e)==size(s)==size(m)

Return:

matrix J; Jordan matrix with list(e,s,m)==jordan(J)

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
ideal e=ideal(2,3);
intvec s=1,2;
intvec m=1,1;
print(jordanmatrix(list(e,s,m)));
↦ 2,0,0,
↦ 0,3,0,
↦ 0,1,3
```

D.3.2.25 jordannf

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 987).

Usage: jordannf(M); matrix M

Assume: eigenvalues of M in basefield

Return: matrix J; Jordan normal form of M

Example:

```
LIB "linalg.lib";
ring R=0,x,dp;
```

```
matrix M[3][3]=3,2,1,0,2,1,0,0,3;
print(M);
↦ 3,2,1,
↦ 0,2,1,
↦ 0,0,3
print(jordannf(M));
↦ 2,0,0,
↦ 0,3,0,
↦ 0,1,3
```

D.4 Commutative algebra

D.4.1 absfact_lib

Library: absfact.lib

Purpose: Absolute factorization for characteristic 0

Authors: Wolfram Decker, decker at math.uni-sb.de
Gregoire Lecerf, lecerf at math.uvsq.fr
Gerhard Pfister, pfister at mathematik.uni-kl.de Martin Lee, mlee at mathematik.uni-kl.de

Overview: A library for computing the absolute factorization of multivariate polynomials f with coefficients in a field K of characteristic zero. Using Trager's idea, the implemented algorithm computes an absolutely irreducible factor by factorizing over some finite extension field L (which is chosen such that V(f) has a smooth point with coordinates in L). Then a minimal extension field is determined making use of the Rothstein-Trager partial fraction decomposition algorithm. absFactorizeBCG uses the algorithm of Bertone, Cheze and Galligo for bivariate polynomials and similar ideas as above to reduce to this case.

References:
G. Cheze, G. Lecerf: Lifting and recombination techniques for absolute factorization. Journal of Complexity, 23(3):380-420, 2007. C. Bertone, G. Cheze, and A. Galligo: Modular las vegas algorithms for polynomial absolute factorization. J. Symb. Comput., 45(12):1280-1295, December 2010

Procedures: See also: Section 5.1.36 [factorize], page 180.

D.4.1.1 absFactorize

Procedure from library `absfact.lib` (see Section D.4.1 [absfact_lib], page 1001).

Usage: absFactorize(p [,s]); p poly, s string

Assume: coefficient field is the field of rational numbers or a transcendental extension thereof

Return: ring `R` which is obtained from the current basering by adding a new parameter (if a string `s` is given as a second input, the new parameter gets this string as name). The ring `R` comes with a list `absolute_factors` with the following entries:

absolute_factors[1]: ideal (the absolute factors)
absolute_factors[2]: intvec (the multiplicities)
absolute_factors[3]: ideal (the minimal polynomials)
absolute_factors[4]: int (total number of nontriv. absolute factors)

The entry `absolute_factors[1][1]` is a constant, the entry `absolute_factors[3][1]` is the parameter added to the current ring.
Each of the remaining entries `absolute_factors[1][j]` stands for a class of conjugated absolute factors. The corresponding entry `absolute_factors[3][j]` is the minimal polynomial of the field extension over which the factor is minimally defined (its degree is the number of conjugates in the class). If the entry `absolute_factors[3][j]` coincides with `absolute_factors[3][1]`, no field extension was necessary for the `jth` (class of) absolute factor(s).

Note: All factors are presented denominator- and content-free. The constant factor (first entry) is chosen such that the product of all (!) the (denominator- and content-free) absolute factors of `p` equals `p / absolute_factors[1][1]`.

Example:

```
LIB "absfact.lib";
ring R = (0), (x,y), lp;
poly p = (-7*x^2 + 2*x*y^2 + 6*x + y^4 + 14*y^2 + 47)*(5x2+y2)^3*(x-y)^4;
def S = absFactorize(p) ;
↦
↦ // 'absFactorize' created a ring, in which a list absolute_factors (the
↦ // absolute factors) is stored.
↦ // To access the list of absolute factors, type (if the name S was assign\
   ed
↦ // to the return value):
↦ //        setring(S); absolute_factors;
↦
setring(S);
absolute_factors;
↦ [1]:
↦    _[1]=1/21125
↦    _[2]=(-14a+19)*x+13*y2+(-7a+94)
↦    _[3]=5*x+(a)*y
↦    _[4]=x-y
↦ [2]:
↦    1,1,3,4
↦ [3]:
↦    _[1]=(a)
↦    _[2]=(7a2-6a-47)
↦    _[3]=(a2+5)
↦    _[4]=(a)
↦ [4]:
↦    12
```

See also: Section D.4.26.20 [absPrimdecGTZ], page 1250; Section 5.1.36 [factorize], page 180.

D.4.1.2 absFactorizeBCG

Procedure from library `absfact.lib` (see Section D.4.1 [absfact_lib], page 1001).

Usage: absFactorizeBCG(p [,s]); p poly, s string

Assume: coefficient field is the field of rational numbers or a transcendental extension thereof

Return: ring `R` which is obtained from the current basering by adding a new parameter (if a string `s` is given as a second input, the new parameter gets this string as name). The ring `R` comes with a list `absolute_factors` with the following entries:

absolute_factors[1]: ideal (the absolute factors)
absolute_factors[2]: intvec (the multiplicities)
absolute_factors[3]: ideal (the minimal polynomials)
absolute_factors[4]: int (total number of nontriv. absolute factors)

The entry `absolute_factors[1][1]` is a constant, the entry `absolute_factors[3][1]` is the parameter added to the current ring.
Each of the remaining entries `absolute_factors[1][j]` stands for a class of conjugated absolute factors. The corresponding entry `absolute_factors[3][j]` is the minimal polynomial of the field extension over which the factor is minimally defined (its degree is the number of conjugates in the class). If the entry `absolute_factors[3][j]` coincides with `absolute_factors[3][1]`, no field extension was necessary for the `jth` (class of) absolute factor(s).

Note: All factors are presented denominator- and content-free. The constant factor (first entry) is chosen such that the product of all (!) the (denominator- and content-free) absolute factors of `p` equals `p / absolute_factors[1][1]`.

Example:

```
LIB "absfact.lib";
ring R = (0), (x,y), lp;
poly p = (-7*x^2 + 2*x*y^2 + 6*x + y^4 + 14*y^2 + 47)*(5x2+y2)^3*(x-y)^4;
def S = absFactorizeBCG(p) ;
↦
↦ // 'absFactorizeBCG' created a ring, in which a list absolute_factors (th\
  e
↦ // absolute factors) is stored.
↦ // To access the list of absolute factors, type (if the name S was assign\
  ed
↦ // to the return value):
↦ //        setring(S); absolute_factors;
↦
setring(S);
absolute_factors;
↦ [1]:
↦    _[1]=1
↦    _[2]=-x+y
↦    _[3]=(-a)*x+y
↦    _[4]=(2a-13)*x+y2+(a)
↦ [2]:
↦    1,4,3,1
↦ [3]:
↦    _[1]=(a)
↦    _[2]=(a)
↦    _[3]=(a2+5)
↦    _[4]=(a2-14a+47)
↦ [4]:
↦    12
```

See also: Section D.4.1.1 [absFactorize], page 1001; Section D.4.26.20 [absPrimdecGTZ], page 1250; Section 5.1.36 [factorize], page 180.

D.4.2 algebra_lib

Library: algebra.lib

Purpose: Compute with Algbras and Algebra Maps

Authors: Gert-Martin Greuel, greuel@mathematik.uni-kl.de,
Agnes Eileen Heydtmann, agnes@math.uni-sb.de,
Gerhard Pfister, pfister@mathematik.uni-kl.de

Procedures:

D.4.2.1 algebra_containment

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: algebra_containment(p,A[,k]); p poly, A ideal, k integer.
A = A[1],...,A[m] generators of subalgebra of the basering

Return:

- k=0 (or if k is not given) an integer:
  1 : if p is contained in the subalgebra K[A[1],...,A[m]]
  0 : if p is not contained in K[A[1],...,A[m]]
- k=1 : a list, say l, of size 2, l[1] integer, l[2] ring, satisfying
  l[1]=1 if p is in the subalgebra K[A[1],...,A[m]] and then the ring
  l[2]: ring, contains poly check = h(y(1),...,y(m)) if p=h(A[1],...,A[m])
  l[1]=0 if p is not in the subalgebra K[A[1],...,A[m]] and then
  l[2] contains the poly check = h(x,y(1),...,y(m)) if p satisfies
  the nonlinear relation p = h(x,A[1],...,A[m]) where
  x = x(1),...,x(n) denote the variables of the basering

Display: if k=0 and printlevel >= voice+1 (default) display the polynomial check

Note: The proc inSubring uses a different algorithm which is sometimes faster.

Theory: The ideal of algebraic relations of the algebra generators A[1],..., A[m] is computed introducing new variables y(i) and the product order with x(i) >> y(i).
p reduces to a polynomial only in the y(i) <=> p is contained in the subring generated by the polynomials A[1],...,A[m].

Example:

```
LIB "algebra.lib";
int p = printlevel; printlevel = 1;
ring R = 0,(x,y,z),dp;
ideal A=x2+y2,z2,x4+y4,1,x2z-1y2z,xyz,x3y-1xy3;
poly p1=z;
poly p2=
x10z3-x8y2z3+2x6y4z3-2x4y6z3+x2y8z3-y10z3+x6z4+3x4y2z4+3x2y4z4+y6z4;
algebra_containment(p1,A);
↦ // x(3)
↦ 0
algebra_containment(p2,A);
↦ // y(1)*y(2)*y(5)^2+y(3)*y(5)^3+4*y(1)*y(2)*y(6)^2+4*y(6)^3*y(7)+2*y(2)*y\
   (5)*y(7)^2
↦ 1
list L = algebra_containment(p2,A,1);
↦
↦ // 'algebra_containment' created a ring as 2nd element of the list.
↦ // The ring contains the polynomial check which defines the algebraic rel\
   ation.
```

```
↦ // To access to the ring and see check you must give the ring a name,
↦ // e.g.:
↦                def S = l[2]; setring S; check;
↦
L[1];
↦ 1
def S = L[2]; setring S;
check;
↦ y(1)*y(2)*y(5)^2+y(3)*y(5)^3+4*y(1)*y(2)*y(6)^2+4*y(6)^3*y(7)+2*y(2)*y(5)\
   *y(7)^2
printlevel = p;
```

D.4.2.2 module_containment

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: module_containment(p,P,M[,k]); p poly, P ideal, M ideal, k int
P = P[1],...,P[n] generators of a subalgebra of the basering,
M = M[1],...,M[m] generators of a module over the subalgebra K[P]

Assume: ncols(P) = nvars(basering), the P[i] are algebraically independent

Return:

- k=0 (or if k is not given), an integer:
  1 : if p is contained in the module <M[1],...,M[m]> over K[P]
  0 : if p is not contained in <M[1],...,M[m]>
- k=1, a list, say l, of size 2, l[1] integer, l[2] ring:
  l[1]=1 : if p is in <M[1],...,M[m]> and then the ring l[2] contains
    the polynomial check = h(y(1),...,y(m),z(1),...,z(n)) if
    p = h(M[1],...,M[m],P[1],...,P[n])
  l[1]=0 : if p is in not in <M[1],...,M[m]>, then l[2] contains the
    poly check = h(x,y(1),...,y(m),z(1),...,z(n)) if p satisfies
    the nonlinear relation p = h(x,M[1],...,M[m],P[1],...,P[n]) where
    x = x(1),...,x(n) denote the variables of the basering

Display: the polynomial h(y(1),...,y(m),z(1),...,z(n)) if k=0, resp. a comment how to access the relation check if k=1, provided printlevel >= voice+1 (default).

Theory: The ideal of algebraic relations of all the generators p1,...,pn, s1,...,st given by P and S is computed introducing new variables y(j), z(i) and the product order: x^a*y^b*z^c > x^d*y^e*z^f if x^a > x^d with respect to the lp ordering or else if z^c > z^f with respect to the dp ordering or else if y^b > y^e with respect to the lp ordering again. p reduces to a polynomial only in the y(j) and z(i), linear in the z(i) <=> p is contained in the module.

Example:

```
LIB "algebra.lib";
int p = printlevel; printlevel = 1;
ring R=0,(x,y,z),dp;
ideal P = x2+y2,z2,x4+y4;           //algebra generators
ideal M = 1,x2z-1y2z,xyz,x3y-1xy3;  //module generators
poly p1=
x10z3-x8y2z3+2x6y4z3-2x4y6z3+x2y8z3-y10z3+x6z4+3x4y2z4+3x2y4z4+y6z4;
module_containment(p1,P,M);
↦ // y(2)*z(2)*z(3)^2+z(1)^3*z(2)^2
↦ 1
poly p2=z;
list l = module_containment(p2,P,M,1);
↦
↦ // 'module_containment' created a ring as 2nd element of the list. The
↦ // ring contains the polynomial check which defines the algebraic relatio\
   n
↦ // for p. To access to the ring and see check you must give the ring
↦ // a name, e.g.:
↦       def S = l[2]; setring S; check;
↦
l[1];
↦ 0
def S = l[2]; setring S; check;
↦ x(3)
printlevel=p;
```

D.4.2.3 inSubring

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: inSubring(p,i); p poly, i ideal

Return:

a list l of size 2, l[1] integer, l[2] string
l[1]=1 if and only if p is in the subring generated by i=i[1],...,i[k],
and then l[2] = y(0)-h(y(1),...,y(k)) if p = h(i[1],...,i[k])
l[1]=0 if and only if p is in not the subring generated by i,
and then l[2] = h(y(0),y(1),...,y(k)) where p satisfies the
nonlinear relation h(p,i[1],...,i[k])=0.

Note: the proc algebra_containment tests the same using a different algorithm, which is often faster
if l[1] == 0 then l[2] may contain more than one relation h(y(0),y(1),...,y(k)), separated by comma

Example:

```
LIB "algebra.lib";
ring q=0,(x,y,z,u,v,w),dp;
poly p=xyzu2w-1yzu2w2+u4w2-1xu2vw+u2vw2+xyz-1yzw+2u2w-1xv+vw+2;
ideal I =x-w,u2w+1,yz-v;
inSubring(p,I);
↦ [1]:
↦    1
↦ [2]:
↦    y(0)-y(1)*y(2)*y(3)-y(2)^2-1
```

D.4.2.4 algDependent

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: algDependent(f[,c]); f ideal (say, f = f1,...,fm), c integer

Return:

a list l of size 2, l[1] integer, l[2] ring:
- l[1] = 1 if f1,...,fm are algebraic dependent, 0 if not
- l[2] is a ring with variables x(1),...,x(n),y(1),...,y(m) if the
  basering has n variables. It contains the ideal 'ker', depending
  only on the y(i) and generating the algebraic relations between the
  f[i], i.e. substituting y(i) by fi yields 0. Of course, ker is
  nothing but the kernel of the ring map
     K[y(1),...,y(m)] —> basering, y(i) -> fi.

Note: Three different algorithms are used depending on c = 1,2,3. If c is not given or c=0, a heuristically best method is chosen. The basering may be a quotient ring.
To access to the ring l[2] and see ker you must give the ring a name, e.g. def S=l[2]; setring S; ker;

Display: The above comment is displayed if printlevel >= 0 (default).

Example:

```
LIB "algebra.lib";
int p = printlevel; printlevel = 1;
ring R = 0,(x,y,z,u,v,w),dp;
ideal I = xyzu2w-1yzu2w2+u4w2-1xu2vw+u2vw2+xyz-1yzw+2u2w-1xv+vw+2,
x-w, u2w+1, yz-v;
list l = algDependent(I);
↦
↦ // The 2nd element of the list l is a ring with variables x(1),...,x(n),
↦ // and y(1),...,y(m) if the basering has n variables and if the ideal
↦ // is f[1],...,f[m]. The ring contains the ideal ker which depends only
↦ // on the y(i) and generates the relations between the f[i].
↦ // I.e. substituting y(i) by f[i] yields 0.
↦ // To access to the ring and see ker you must give the ring a name,
↦ // e.g.:
↦              def S = l[2]; setring S; ker;
↦
l[1];
↦ 1
def S = l[2]; setring S;
ker;
↦ ker[1]=y(2)*y(3)*y(4)+y(3)^2-y(1)+1
printlevel = p;
```

D.4.2.5 alg_kernel

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: alg_kernel(phi,pr[,s,c]); phi map to basering, pr preimage ring, s string (name of kernel in pr), c integer.

Return: a string, the kernel of phi as string.
If, moreover, a string s is given, the algorithm creates, in the preimage ring pr the kernel of phi with name s.
Three different algorithms are used depending on c = 1,2,3. If c is not given or c=0, a heuristically best method is chosen. (algorithm 1 uses the preimage command)

Note: Since the kernel of phi lives in pr, it cannot be returned to the basering. If s is given, the user has access to it in pr via s. The basering may be a quotient ring.

Example:

```
LIB "algebra.lib";
ring r = 0,(a,b,c),ds;
ring s = 0,(x,y,z,u,v,w),dp;
ideal I = x-w,u2w+1,yz-v;
map phi = r,I;                // a map from r to s:
alg_kernel(phi,r);            // a,b,c ---> x-w,u2w+1,yz-v
↦ 0
ring S = 0,(a,b,c),ds;
ring R = 0,(x,y,z),dp;
qring Q = std(x-y);
ideal i = x, y, x2-y3;
map phi = S,i;                 // a map to a quotient ring
alg_kernel(phi,S,"ker",3);     // uses algorithm 3
↦ a-b,b^3-b^2+c
setring S;                     // you have access to kernel in preimage
ker;
↦ ker[1]=a-b
↦ ker[2]=c-b2+b3
```

D.4.2.6 is_injective

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: is_injective(phi,pr[,c,s]); phi map, pr preimage ring, c int, s string

Return:

- 1 (type int) if phi is injective, 0 if not (if s is not given).
- If s is given, return a list l of size 2, l[1] int, l[2] ring:
  l[1] is 1 if phi is injective, 0 if not
  l[2] is a ring with variables x(1),...,x(n),y(1),...,y(m) if the
  basering has n variables and the map m components, it contains the
  ideal 'ker', depending only on the y(i), the kernel of the given map

Note: Three different algorithms are used depending on c = 1,2,3. If c is not given or c=0, a heuristically best method is chosen. The basering may be a quotient ring. However, if the preimage ring is a quotient ring, say pr = P/I, consider phi as a map from P and then the algorithm returns 1 if the kernel of phi is 0 mod I. To access to the ring l[2] and see ker you must give the ring a name, e.g. def S=l[2]; setring S; ker;

Display: The above comment is displayed if printlevel >= 0 (default).

Example:

```
LIB "algebra.lib";
int p = printlevel;
ring r = 0,(a,b,c),ds;
ring s = 0,(x,y,z,u,v,w),dp;
ideal I = x-w,u2w+1,yz-v;
map phi = r,I;                    // a map from r to s:
is_injective(phi,r);              // a,b,c ---> x-w,u2w+1,yz-v
↦ 1
ring R = 0,(x,y,z),dp;
ideal i = x, y, x2-y3;
map phi = R,i;                    // a map from R to itself, z --> x2-y3
list l = is_injective(phi,R,"");
```

```
↦
↦ // The 2nd element of the list is a ring with variables x(1),...,x(n),
↦ // y(1),...,y(m) if the basering has n variables and the map is
↦ // F[1],...,F[m].
↦ // It contains the ideal ker, the kernel of the given map y(i) --> F[i].
↦ // To access to the ring and see ker you must give the ring a name,
↦ // e.g.:
↦      def S = l[2]; setring S; ker;
↦
l[1];
↦ 0
def S = l[2]; setring S;
ker;
↦ ker[1]=y(2)^3-y(1)^2+y(3)
```

D.4.2.7 is_surjective

Procedure from library algebra.lib (see Section D.4.2 [algebra_lib], page 1003).

Usage: is_surjective(phi); phi map to basering, or ideal defining it

Return: an integer, 1 if phi is surjective, 0 if not

Note: The algorithm returns 1 if and only if all the variables of the basering are contained in the polynomial subalgebra generated by the polynomials defining phi. Hence, it tests surjectivity in the case of a global odering. If the basering has local or mixed ordering or if the preimage ring is a quotient ring (in which case the map may not be well defined) then the return value 1 needs to be interpreted with care.

Example:

```
LIB "algebra.lib";
ring R = 0,(x,y,z),dp;
ideal i = x, y, x2-y3;
map phi = R,i;                    // a map from R to itself, z->x2-y3
is_surjective(phi);
↦ 0
qring Q = std(ideal(z-x37));
map psi = R, x,y,x2-y3;           // the same map to the quotient ring
is_surjective(psi);
↦ 1
ring S = 0,(a,b,c),dp;
map psi = R,ideal(a,a+b,c-a2+b3); // a map from R to S,
is_surjective(psi);               // x->a, y->a+b, z->c-a2+b3
↦ 1
```

D.4.2.8 is_bijective

Procedure from library algebra.lib (see Section D.4.2 [algebra_lib], page 1003).

Usage: is_bijective(phi,pr); phi map to basering, pr preimage ring

Return: an integer, 1 if phi is bijective, 0 if not

Note: The algorithm checks first injectivity and then surjectivity. To interpret this for local/mixed orderings, or for quotient rings type help is_surjective; and help is_injective;

Display: A comment if printlevel >= voice-1 (default)

Example:

```
LIB "algebra.lib";
int p = printlevel;  printlevel = 1;
ring R = 0,(x,y,z),dp;
ideal i = x, y, x2-y3;
map phi = R,i;                       // a map from R to itself, z->x2-y3
is_bijective(phi,R);
↦ // map not injective
↦ 0
qring Q = std(z-x2+y3);
is_bijective(ideal(x,y,x2-y3),Q);
↦ 1
ring S = 0,(a,b,c,d),dp;
map psi = R,ideal(a,a+b,c-a2+b3,0); // a map from R to S,
is_bijective(psi,R);                 // x->a, y->a+b, z->c-a2+b3
↦ // map injective, but not surjective
↦ 0
qring T = std(d,c-a2+b3);
↦ // ** _ is no standard basis
map chi = Q,a,b,a2-b3;               // amap between two quotient rings
is_bijective(chi,Q);
↦ 1
printlevel = p;
```

D.4.2.9 noetherNormal

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: noetherNormal(id[,p]); id ideal, p integer

Return:

a list l of two ideals, say I,J:
- I defines a map (coordinate change in the basering), such that:
- J is generated by a subset of the variables with size(J) = dim(id)
  if we define map phi=basering,I;
  then k[var(1),...,var(n)]/phi(id) is finite over k[J].
If p is given, 0<=p<=100, a sparse coordinate change with p percent
of the matrix entries being 0 (default: p=0 i.e. dense)

Note: Designed for characteristic 0.It works also in char k > 0 if it terminates,but may result in an infinite loop in small characteristic.

Example:

```
LIB "algebra.lib";
ring r=0,(x,y,z),dp;
ideal i= xy,xz;
noetherNormal(i);
↦ [1]:
↦    _[1]=x
↦    _[2]=2x+y
↦    _[3]=3x+4y+z
↦ [2]:
↦    _[1]=y
↦    _[2]=z
```

D.4.2.10 mapIsFinite

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: mapIsFinite(phi,R[,J]); R the preimage ring of the map phi: R —> basering
J an ideal in the basering, J = 0 if not given

Return: 1 if R —> basering/J is finite and 0 else

Note: R may be a quotient ring (this will be ignored since a map R/I–>S/J is finite if and only if the induced map R–>S/J is finite).

Example:

```
LIB "algebra.lib";
ring r = 0,(a,b,c),dp;
ring s = 0,(x,y,z),dp;
ideal i= xy;
map phi= r,(xy)^3+x2+z,y2-1,z3;
mapIsFinite(phi,r,i);
↦ 1
```

See also: Section D.4.2.11 [finitenessTest], page 1011.

D.4.2.11 finitenessTest

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: finitenessTest(J[,v]); J ideal, v intvec (say v1,...,vr with vi>0)

Return:

a list l with l[1] integer, l[2], l[3], l[4] ideals
- l[1] = 1 if var(v1),...,var(vr) are in l[2] and 0 else
- l[2] (resp. l[3]) contains those variables which occur,
  (resp. do not occur) as pure power in the leading term of one of the
  generators of J,
- l[4] contains those J[i] for which the leading term is a pure power
  of a variable (which is then in l[2])
(default: v = [1,2,..,nvars(basering)])

Theory: If J is a standard basis of an ideal generated by x_1 - f_1(y),..., x_n - f_n with y_j ordered lexicographically and y_j >> x_i, then, if y_i appears as pure power in the leading term of J[k], J[k] defines an integral relation for y_i over the y_(i+1),... and the f's. Moreover, in this situation, if l[2] = y_1,...,y_r, then K[y_1,...y_r] is finite over K[f_1..f_n]. If J contains furthermore polynomials h_j(y), then K[y_1,...y_z]/<h_j> is finite over K[f_1..f_n]. For a proof cf. Prop. 3.1.5, p. 214. in [G.-M. Greuel, G. Pfister: A SINGULAR Introduction to Commutative Algebra, 2nd Edition, Springer Verlag (2007)]

Example:

```
LIB "algebra.lib";
ring s = 0,(x,y,z,a,b,c),(lp(3),dp);
ideal i= a -(xy)^3+x2-z, b -y2-1, c -z3;
ideal j = a -(xy)^3+x2-z, b -y2-1, c -z3, xy;
finitenessTest(std(i),1..3);
↦ [1]:
↦    0
↦ [2]:
↦    _[1]=y
↦     _[2]=z
↦  [3]:
↦     _[1]=x
↦     _[2]=a
↦     _[3]=b
↦     _[4]=c
↦  [4]:
↦     _[1]=z3-c
↦     _[2]=y2-b+1
finitenessTest(std(j),1..3);
↦  [1]:
↦     1
↦  [2]:
↦     _[1]=x
↦     _[2]=y
↦     _[3]=z
↦  [3]:
↦     _[1]=a
↦     _[2]=b
↦     _[3]=c
↦  [4]:
↦     _[1]=z3-c
↦     _[2]=y2-b+1
↦     _[3]=x2-z+a
```

D.4.2.12 nonZeroEntry

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 1003).

Usage: nonZeroEntry(id); id=object for which the test 'id[i]!=0', i=1,..,N, N=size(id) (resp. ncols(id) for id of type ideal or module) is defined (e.g. ideal, vector, list of polynomials, intvec,...)

Return:

a list, say l, with l[1] an integer, l[2], l[3] integer vectors:
- l[1] number of non-zero entries of id
- l[2] intvec of size l[1] with l[2][i]=i if id[i] != 0
  in case l[1]!=0 (and l[2]=0 if l[1]=0)
- l[3] intvec with l[3][i]=1 if id[i]!=0 and l[3][i]=0 else

Note:

Example:

```
LIB "algebra.lib";
ring r = 0,(a,b,c),dp;
poly f = a3c+b3+c2+a;
intvec v = leadexp(f);
nonZeroEntry(v);
↦  [1]:
↦     2
↦  [2]:
↦     1,3
↦  [3]:
↦     1,0,1
intvec w;
```

```
list L = 37,0,f,v,w;
nonZeroEntry(L);
↦ [1]:
↦    3
↦ [2]:
↦    1,3,4
↦ [3]:
↦    1,0,1,1,0
```

D.4.3 assprimeszerodim_lib

Library: assprimeszerodim.lib

Purpose: associated primes of a zero-dimensional ideal

Authors: N. Idrees nazeranjawwad@gmail.com
G. Pfister pfister@mathematik.uni-kl.de
A. Steenpass steenpass@mathematik.uni-kl.de
S. Steidel steidel@mathematik.uni-kl.de

Overview: A library for computing the associated primes and the radical of a zero-dimensional ideal in the polynomial ring over the rational numbers, Q[x_1,...,x_n], using modular computations.

Procedures: See also: Section D.4.26 [primdec_lib], page 1239.

D.4.3.1 zeroRadical

Procedure from library `assprimeszerodim.lib` (see Section D.4.3 [assprimeszerodim_lib], page 1013).

Usage: zeroRadical(I[, exactness]); I ideal, exactness int

Assume: I is zero-dimensional in Q[variables]

Return: the radical of I

Note: A final test is applied to the result if exactness != 0 (default), otherwise no final test is done.

Example:

```
LIB "assprimeszerodim.lib";
ring R = 0, (x,y), dp;
ideal I = xy4-2xy2+x, x2-x, y4-2y2+1;
zeroRadical(I);
↦ _[1]=y2-1
↦ _[2]=x2-x
```

D.4.3.2 assPrimes

Procedure from library `assprimeszerodim.lib` (see Section D.4.3 [assprimeszerodim_lib], page 1013).

Assume: I is zero-dimensional over Q[variables]

Return: a list of the associated primes of I

Note: A final test is applied to the result if exactness != 0 (default), otherwise no final test is done.

Example:

```
LIB "assprimeszerodim.lib";
ring R = 0,(a,b,c,d,e,f),dp;
ideal I =
2fb+2ec+d2+a2+a,
2fc+2ed+2ba+b,
2fd+e2+2ca+c+b2,
2fe+2da+d+2cb,
f2+2ea+e+2db+c2,
2fa+f+2eb+2dc;
assPrimes(I);
↦ [1]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=49a-75b-94c+11d+94e+f
↦ [2]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=49a-75b-94c+11d+94e+f-2
↦ [3]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=49a-75b-94c+11d+94e+f+19
↦ [4]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=49a-75b-94c+11d+94e+f+30
↦   [5]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=49a-75b-94c+11d+94e+f+49
↦   [6]:
```

```
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=49a-75b-94c+11d+94e+f+51
↦  [7]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f-7
↦  [8]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=147a-225b-282c+33d+282e+3f+1
↦   [9]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=147a-225b-282c+33d+282e+3f+49
↦ [10]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=147a-225b-282c+33d+282e+3f+50
↦ [11]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f+56
↦ [12]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f+91
↦  [13]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f+97
↦  [14]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f+98
↦  [15]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=147a-225b-282c+33d+282e+3f+146
↦   [16]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=147a-225b-282c+33d+282e+3f+154
↦  [17]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-61386ac-39066ad-57966bd-6840d2-25380ae-36096be-67422ce+28254de+\
   9633e2+6498af-14856bf+21486cf-33684df-41736ef-26505f2-3087a+4725b-24771c-\
   20226d-18612e+3186f+2032
↦   [18]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-61386ac-39066ad-57966bd-6840d2-25380ae-36096be-67422ce+28254de+\
   9633e2+6498af-14856bf+21486cf-33684df-41736ef-26505f2-3087a+4725b-24771c-\
   20226d-18612e+3186f+9424
↦   [19]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-61386ac-39066ad-57966bd-6840d2-25380ae-36096be-67422ce+28254de+\
   9633e2+6498af-14856bf+21486cf-33684df-41736ef-26505f2+3087a-4725b-36615c-\
   18840d-6768e+3312f+5119
↦  [20]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-61386ac-39066ad-57966bd-6840d2-25380ae-36096be-67422ce+28254de+\
   9633e2+6498af-14856bf+21486cf-33684df-41736ef-26505f2+3087a-4725b-36615c-\
   18840d-6768e+3312f+12511
↦  [21]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-81848ac-52088ad-77288bd-9120d2-33840ae-48128be-89896ce+37672de+\
   12844e2+8664af-19808bf+28648cf-44912df-55648ef-35340f2-3136a+4800b-34908c\
   -26748d-22936e+4268f+6897
↦  [22]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-81848ac-52088ad-77288bd-9120d2-33840ae-48128be-89896ce+37672de+\
   12844e2+8664af-19808bf+28648cf-44912df-55648ef-35340f2+3136a-4800b-46940c\
   -25340d-10904e+4396f+10033
↦ [23]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-184158ac-117198ad-173898bd-20520d2-76140ae-108288be-202266ce+84\
   762de+28899e2+19494af-44568bf+64458cf-101052df-125208ef-79515f2-7203a+110\
   25b-78261c-60216d-51888e+9600f+6733
↦ [24]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-184158ac-117198ad-173898bd-20520d2-76140ae-108288be-202266ce+84\
   762de+28899e2+19494af-44568bf+64458cf-101052df-125208ef-79515f2-7203a+110\
   25b-78261c-60216d-51888e+9600f+28909
↦ [25]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-184158ac-117198ad-173898bd-20520d2-76140ae-108288be-202266ce+84\
   762de+28899e2+19494af-44568bf+64458cf-101052df-125208ef-79515f2+7203a-110\
   25b-105897c-56982d-24252e+9894f+13936
↦ [26]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-184158ac-117198ad-173898bd-20520d2-76140ae-108288be-202266ce+84\
   762de+28899e2+19494af-44568bf+64458cf-101052df-125208ef-79515f2+7203a-110\
   25b-105897c-56982d-24252e+9894f+36112
↦  [27]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-245544ac-156264ad-231864bd-27360d2-101520ae-144384be-269688ce+1\
   13016de+38532e2+25992af-59424bf+85944cf-134736df-166944ef-106020f2-9408a+\
   14400b-104724c-80244d-68808e+12804f+6403
```

```
↦ [28]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-245544ac-156264ad-231864bd-27360d2-101520ae-144384be-269688ce+1\
   13016de+38532e2+25992af-59424bf+85944cf-134736df-166944ef-106020f2+9408a-\
   14400b-140820c-76020d-32712e+13188f+15811
↦ [29]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
  39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-940\
  80a+144000b-187836c-255516d-332760e+37068f+52441
↦ [30]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-858\
   48a+131400b-203628c-253668d-316968e+37236f+52273
↦  [31]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-376\
   32a+57600b-296124c-242844d-224472e+38220f+16297
↦  [32]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-364\
   56a+55800b-298380c-242580d-222216e+38244f+16633
↦  [33]:
↦     _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-294\
   00a+45000b-311916c-240996d-208680e+38388f+18817
↦   [34]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦    _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦    _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2-199\
   92a+30600b-329964c-238884d-190632e+38580f+65041
↦ [35]:
↦    _[1]=cd+be+af+1/2f
↦    _[2]=c2+2bd+2ae+f2+e
↦    _[3]=bc+ad+ef+1/2d
↦    _[4]=b2+2ac+e2+2df+c
↦    _[5]=ab+de+cf+1/2b
↦    _[6]=a2+d2+2ce+2bf+a
↦    _[7]=de2+d2f+2cef+bf2-5/36f
↦    _[8]=d2e+ce2-af2-1/2f2-1/9e
↦    _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦    _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦    _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦    _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦    _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦    _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦    _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦    _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦    _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦    _[18]=acef+1/2cef-1/36af-1/72f
↦    _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦    _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦    _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦    _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦    _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
  39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+199\
  92a-30600b-406668c-229908d-113928e+39396f+85033
↦ [36]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+294\
   00a-45000b-424716c-227796d-95880e+39588f+48217
↦  [37]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+364\
   56a-55800b-438252c-226212d-82344e+39732f+53089
↦  [38]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦      _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦      _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦      _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦      _[18]=acef+1/2cef-1/36af-1/72f
↦      _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦      _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦      _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦      _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦      _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦      _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦      _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦      _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦      _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦      _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦      _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦      _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦      _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦      _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦      _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦      _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦      _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦      _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦      _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦      _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦      _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+376\
   32a-57600b-440508c-225948d-80088e+39756f+53929
↦  [39]:
↦      _[1]=cd+be+af+1/2f
↦      _[2]=c2+2bd+2ae+f2+e
↦      _[3]=bc+ad+ef+1/2d
↦      _[4]=b2+2ac+e2+2df+c
↦      _[5]=ab+de+cf+1/2b
↦      _[6]=a2+d2+2ce+2bf+a
↦      _[7]=de2+d2f+2cef+bf2-5/36f
↦      _[8]=d2e+ce2-af2-1/2f2-1/9e
↦      _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦      _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦      _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦      _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦      _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦      _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦     _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦     _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦     _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦     _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦     _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦     _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦     _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦     _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦     _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦     _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦     _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦     _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦     _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦     _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+858\
   48a-131400b-533004c-215124d+12408e+40740f+138121
↦  [40]:
↦     _[1]=cd+be+af+1/2f
↦     _[2]=c2+2bd+2ae+f2+e
↦     _[3]=bc+ad+ef+1/2d
↦     _[4]=b2+2ac+e2+2df+c
↦     _[5]=ab+de+cf+1/2b
↦     _[6]=a2+d2+2ce+2bf+a
↦     _[7]=de2+d2f+2cef+bf2-5/36f
↦     _[8]=d2e+ce2-af2-1/2f2-1/9e
↦     _[9]=bde+ae2+adf+ef2+1/2e2+1/2df+1/18c
↦     _[10]=ade+acf+e2f+df2+1/2de+1/2cf+1/18b
↦     _[11]=d3-3be2-6aef-f3-3ef-1/4d
↦     _[12]=bd2-2acf-e2f-df2-cf-1/12b
↦     _[13]=ad2+2ace+e3+4def+cf2+1/2d2+ce
↦     _[14]=e3f+3def2+cf3+1/36be+1/12af+1/24f
↦     _[15]=ce2f+bef2-1/36ad-1/9ef-1/72d
↦     _[16]=be2f+bdf2+2aef2+1/2f4+ef2-1/36ac-1/72e2-1/72c
↦     _[17]=ae2f+adf2+ef3+1/2e2f+1/2df2+1/36de+1/18cf
↦     _[18]=acef+1/2cef-1/36af-1/72f
↦     _[19]=e4-6d2f2-12cef2-8bf3+1/18bd+2/9ae+13/12f2+1/9e
↦     _[20]=ce3-3aef2-1/2f4-3/2ef2-1/36ac-1/8e2-1/9df-1/72c
↦     _[21]=be3-3adf2-2ef3-3/2df2-5/36cf
↦     _[22]=ae3-3acf2-2df3+1/2e3-3/2cf2+1/18ce-1/9bf
↦     _[23]=ace2+d2f2+3cef2+2bf3+1/2ce2-1/18bd-1/9ae-5/18f2-1/18e
↦     _[24]=e2f3+1/2df4+1/72d2f+1/18cef+5/72bf2-13/2592f
↦     _[25]=def3+1/2cf4-1/72ce2-1/36bef+1/36af2+1/72f2+1/648e
↦    _[26]=cef3+1/2bf4-1/72be2-1/18bdf-1/18aef-1/12f3-1/36ef
↦    _[27]=bef3+1/2af4+1/4f4-1/72ae2-1/36adf-1/18ef2-1/144e2-1/72df-1/1296c
↦    _[28]=aef3+1/5f5+1/2ef3+1/36acf+1/36e2f+1/18df2+1/72cf+1/1620b
↦    _[29]=d2f3+1/2bf4+1/24be2+1/9bdf+1/12aef-1/36f3+1/24ef
↦    _[30]=bdf3+1/5f5-1/18acf-1/36e2f-1/12df2-1/36cf-1/1080b
↦    _[31]=adf3+1/2ef4+1/2df3-1/36ace-1/72e3-1/36def+1/36cf2-1/72ce
↦    _[32]=acf3+1/2df4+1/2cf3-1/72d2f-1/36cef+1/72bf2+5/2592f
↦    _[33]=ef5+5/36def2+5/36cf3-1/1296be+5/1296af+5/2592f
↦    _[34]=df5+5/36d2f2+5/36bf3+1/324bd-5/432f2
↦    _[35]=cf5-5/36bef2+1/1296ad+1/216ef+1/2592d
↦    _[36]=bf5+5/36bdf2-5/72f4-1/432ac-1/864e2-1/162df-1/864c
↦    _[37]=af5+1/2f5+5/36adf2+5/36ef3+5/72df2+1/1296de+1/216cf
↦    _[38]=f7+35/72df4+35/2592d2f+7/288bf2-5/3456f
↦    _[39]=-736632ac-468792ad-695592bd-82080d2-304560ae-433152be-809064ce+3\
   39048de+115596e2+77976af-178272bf+257832cf-404208df-500832ef-318060f2+940\
   80a-144000b-548796c-213276d+28200e+40908f+146521
```

D.4.4 cisimplicial_lib

Library: cisimplicial.lib

Purpose: . Determines if the toric ideal of a simplicial toric variety is a complete intersection

Authors: I.Bermejo, ibermejo@ull.es
I.Garcia-Marco, iggarcia@ull.es

Overview: A library for determining if a simplicial toric ideal is a complete intersection with NO NEED of computing explicitly a system of generators of such ideal. The procedures are based on two papers: I. Bermejo, I. Garcia-Marco and J.J. Salazar-Gonzalez: 'An algorithm for checking whether the toric ideal of an affine monomial curve is a complete intersection', J. Symbolic Computation 42 (2007) pags: 971–991 and I.Bermejo and I. Garcia-Marco: 'Complete intersections in simplicial toric varieties', Preprint (2010)

Procedures:

D.4.4.1 minMult

Procedure from library `cisimplicial.lib` (see Section D.4.4 [cisimplicial_lib], page 1045).

Usage: minMult (a, b); a integer, b integer vector.

Return: an integer k, the minimum positive integer such that k*a belongs to the semigroup generated by the integers in b.

Assume: a is a positive integer, b is a vector of positive integers.

Example:

```
LIB "cisimplicial.lib";
int a = 46;
intvec b = 13,17,59;
minMult(a,b);
↦ 3
"// 3*a = 8*b[1] + 2*b[2]"
```

D.4.4.2 belongSemigroup

Procedure from library `cisimplicial.lib` (see Section D.4.4 [cisimplicial_lib], page 1045).

Usage: belongSemigroup(v,A[,k]); v is an integral vector, A is an integral matrix, n is a positive integer.

Return: counters, a vector with nonnegative entries such that A*counters = v. If it does not exist such a vector, it returns 0. If a third parameter k is introduced, it will only consider the first k columns of A.

Assume: A is a matrix with nonnegative entries, nonzero columns, v is a nonnegative vector and nrows(v) = nrows(A).

Example:

```
LIB "cisimplicial.lib";
intmat A[3][4] = 10,3,2,1,2,1,1,3,5,0,1,2;
print(A);
↦     10     3     2     1
↦      2     1     1     3
↦      5     0     1     2
intvec v = 23,12,10;
belongSemigroup(v,A);
↦ 1,3,1,2
"// A * (1,3,1,2) = v";
↦ // A * (1,3,1,2) = v
belongSemigroup(v,A,3);
↦ 0
"// v is not a combination of the first 3 columns of A";
↦ // v is not a combination of the first 3 columns of A
intvec w = 12,4,1;
belongSemigroup(w,A);
↦ 0
"// w is not a combination of the columns of A";
↦ // w is not a combination of the columns of A
```

D.4.4.3 oneDimBelongSemigroup

Procedure from library `cisimplicial.lib` (see Section D.4.4 [cisimplicial_lib], page 1045).

Usage: oneDimBelongSemigroup(n,v[,m]); v is an integral vector, n is a positive integer[, m is a positive integer].

Return: counters, a vector with nonnegative entries such that v*counters = n. If it does not exist such a vector, it returns 0. If a third parameter m is introduced, it will only consider the first m entries of v.

Assume: v is an integral vector with positive entries.

Example:

```
LIB "cisimplicial.lib";
int a = 95;
intvec v = 18,51,13;
oneDimBelongSemigroup(a,v);
↦ 1,1,2
"// 95 = 1*18 + 1*25 + 2*13";
```

```
↦ // 95 = 1*18 + 1*25 + 2*13
oneDimBelongSemigroup(a,v,2);
↦ 0
"// 95 is not a combination of 18 and 52;";
↦ // 95 is not a combination of 18 and 52;
```

D.4.4.4 cardGroup

Procedure from library `cisimplicial.lib` (see Section D.4.4 [cisimplicial_lib], page 1045).

Usage: cardGroup(A[,n]); A is a matrix with integral coefficients.

Return: It returns a bigint. If we denote by ZA the group generated by the columns of the matrix A, then it returns the number of elements of the group of Z^m / ZA, where m = number of rows of A. If a second parameter n is introduced, it will
only consider the first n columns of A. It returns 0 if Z^m / ZA is infinite; this is, when rank ZA < m.

Example:

```
LIB "cisimplicial.lib";
intmat A[3][5] = 24,  0,  0,  8, 3,
0, 24,  0, 10, 6,
0,  0, 24,  5, 9;
cardGroup(A);
↦ 72
```

D.4.4.5 isCI

Procedure from library `cisimplicial.lib` (see Section D.4.4 [cisimplicial_lib], page 1045).

Usage: isCI(A); A is an integral matrix

Return: 1 if the simplicial toric ideal I(A) is a complete intersection and 0 otherwise. If printlevel > 0 and I(A) is a complete intersection it also shows a minimal set of generators of I(A)

Assume: A is an m x n integral matrix with nonnegative entries and for every 1 <= i <= m, there exist a column in A whose i-th coordinate is not null and the rest are 0.

Example:

```
LIB "cisimplicial.lib";
intmat A[2][5] = 60,0,140,150,21,0,60,140,150,21;
print(A);
↦     60     0   140   150    21
↦      0    60   140   150    21
printlevel = 0;
isCI(A);
↦ // It is a complete intersection
↦ 1
printlevel = 1;
isCI(A);
↦ // Generators of the toric ideal
↦ toric[1]=-x(1)^7*x(2)^7+x(3)^3
↦ toric[2]=x(5)^10-x(1)*x(2)*x(4)
↦ toric[3]=-x(1)^5*x(2)^5+x(4)^2
↦ // It is a complete intersection
↦ 1
```

```
intmat B[3][5] = 12,0,0,1,2,0,10,0,3,2,0,0,8,3,3;
print(B);
↦     12     0     0     1     2
↦      0    10     0     3     2
↦      0     0     8     3     3
isCI(B);
↦  // It is NOT a Complete Intersection.
↦ 0
printlevel=0;
```

D.4.5 curveInv_lib

Library: curveInv.lib

Purpose: A library for computing invariants of curves

Author: Peter Chini, chini@rhrk.uni-kl.de

Overview: This library provides a collection of procedures for computing invariants of curve singularities. Invariants that can be computed are: - the delta invariant
- the multiplicity of the conductor: the length of Normalization(R)/C, where C denotes the conductor
- the Deligne number
- the colength of derivations along the normalization - the length of Der(Normalization(R/I)) / R/I

In addition, it is possible to compute the conductor of a ring S = R/I, where R is a (localized) polynomial ring.

Theory: Computing the Deligne number of curve singularities and an algorithmic framework for differential algebras in SINGULAR;
Chapter 5 - Master's Thesis of Peter Chini - August 2015

Procedures:

D.4.5.1 curveDeltaInv

Procedure from library `curveInv.lib` (see Section D.4.5 [curveInv_lib], page 1048).

Usage: curveDeltaInv(I); I ideal

Assume: I is a radical ideal, dim(R/I) = 1

Return: the delta invariant of R/I

Note: - output -1 means: delta invariant is infinite
- the optional parameter can be used if the normalization has already been computed. If a list L contains the output of the procedure normal (with options prim, wd and usering if the ring has a mixed ordering), apply curveDeltaInv(I,L)

Example:

```
LIB "curveInv.lib";
ring R = 0,(x,y,z),ds;
/////////////////////////////
// Finite delta invariant //
/////////////////////////////
ideal I = x2y-y2z,x2-y2+z2;
```

```
curveDeltaInv(radical(I));
↦ 9
//////////////////////////////
// Infinite delta invariant //
//////////////////////////////
ideal J = xyz;
curveDeltaInv(radical(J));
↦ -1
```

See also: Section D.4.5.2 [curveConductorMult], page 1049; Section D.4.5.3 [curveDeligneNumber], page 1050.

D.4.5.2 curveConductorMult

Procedure from library `curveInv.lib` (see Section D.4.5 [curveInv_lib], page 1048).

Usage: curveConductorMult(I); I ideal

Assume: I is a radical ideal, dim(R/I) = 1

Return: the multiplicity of the conductor

Note: the optional parameter can be used if the normalization has already been computed. If a list L contains the output of the procedure normal (with options prim, wd and usering if the ring has a mixed ordering), apply curveConductorMult(I,L)

Example:

```
LIB "curveInv.lib";
//////////////////////////////////////////////
// Mutltiplicity of the conductor of curves //
//////////////////////////////////////////////
ring R = 0,(x,y,z),ds;
// Example 1:
ideal I = x2-y4z,z3y2+xy2;
I = std(radical(I));
curveConductorMult(I);
↦ 23
// Example 2:
ideal I = x*(y+z)^3 - y3, x2y2 + z5;
↦ // ** redefining I (ideal I = x*(y+z)^3 - y3, x2y2 + z5;) ./examples/curv\
   eConductorMult.sing:11
I = std(radical(I));
curveConductorMult(I);
↦ 19
kill R;
////////////////////////////////////////////////////////
// Mutltiplicity of the conductor of Gorenstein curve //
////////////////////////////////////////////////////////
ring R = 0,(x,y),ds;
ideal I = xy;
// In such a case, the conductor multiplicity c satisfies: c = 2*delta
// Delta invariant:
curveDeltaInv(I);
↦ 1
// Conductor Multiplicity:
curveConductorMult(I);
```

```
↦ 2
```

See also: Section D.4.23.17 [normalConductor], page 1207.

D.4.5.3 curveDeligneNumber

Procedure from library `curveInv.lib` (see Section D.4.5 [curveInv_lib], page 1048).

Usage: curveDeligneNumber(I); I ideal

Assume: I is a radical ideal, dim(R/I) = 1

Return: the Deligne number of R/I

Remarks: The Deligne number e satisfies by definition: e = 3delta - m. So the algorithm splits the computation into two parts: one part computes the delta invariant, the other part the colength of derivations m.

Note: the optional parameter can be used if the normalization has already been computed. If a list L contains the output of the procedure normal (with options prim, wd and usering if the ring has a mixed ordering), apply curveDeligneNumber(I,L)

Example:

```
LIB "curveInv.lib";
//////////////////////////////
// Deligne number of curves //
//////////////////////////////
// Example 1:
ring R = 0,(x,y,z),ds;
ideal I = x2-y4z,z3y2+xy2;
I = std(radical(I));
curveDeligneNumber(I);
↦ 30
// Example 2:
ring S = 0,(x,y),ds;
ideal I = (x+y)*(x2-y3);
curveDeligneNumber(I);
↦ 5
// Example 3:
ideal J = (x2-y3)*(x2+y2)*(x-y);
curveDeligneNumber(J);
↦ 15
// Let us also compute the milnor number of this complete intersection:
milnor(J);
↦ 17
// We see that the Milnor number is bigger than the Deligne number. Hence, this
// curve cannot be quasi homogeneous. This can also be verified by Saitos criterion:
reduce(J[1],std(jacob(J[1])));
↦ -1/5y6+19/50y7
```

See also: Section D.4.5.4 [curveColengthDerivations], page 1050; Section D.4.5.1 [curveDeltaInv], page 1048.

D.4.5.4 curveColengthDerivations

Procedure from library `curveInv.lib` (see Section D.4.5 [curveInv_lib], page 1048).

Usage: curveColengthDerivations(I); I ideal

Assume: I is a radical ideal

Return: the colength of derivations of R/I

Remarks: The procedure goes through all branches and computes the colength of derivations there. Then the d-dimension of the minimal primes is computed. After that, everything is summed up.

Note: the optional parameter can be used if the normalization has already been computed. If a list L contains the output of the procedure normal (with options prim, wd and usering if the ring has a mixed ordering), apply curveColengthDerivations(I,L)

Example:

```
LIB "curveInv.lib";
///////////////////////////////////////
// colength of derivations of curves //
///////////////////////////////////////
// Example 1:
ring R = 0,(x,y,z),ds;
ideal I = x2-y4z,z3y2+xy2;
I = std(radical(I));
curveColengthDerivations(I);
↦ 9
// Example 2:
ring S = 0,(x,y),ds;
ideal I = (x+y)*(x2-y3);
curveColengthDerivations(I);
↦ 4
// Example 3:
ideal J = (x2-y3)*(x2+y2)*(x-y);
curveColengthDerivations(J);
↦ 15
```

D.4.6 decomp_lib

Library: decomp.lib

Purpose: Functional Decomposition of Polynomials

Author: Christian Gorzel, University of Muenster
email: gorzelc@math.uni-muenster.de

Overview: This library implements functional uni-multivariate decomposition of multivariate polynomials.

A (multivariate) polynomial f is a composite if it can be written as

$g \circ h$ where g is univariate and h is multivariate, where $\deg(g), \deg(h) > 1$.

Uniqueness for monic polynomials is up to linear coordinate change $g \circ h = g(x/c - d) \circ c(h(x) + d)$.

If f is a composite, then `decompose(f);` returns an ideal (g,h); such that $\deg(g) < \deg(f)$ is maximal, ($\deg(h) \geq 2$). The polynomial h is, by the maximality of $\deg(g)$, not a composite.

The polynomial g is univariate in the (first) variable vvar of f, such that deg_vvar(f) is maximal.

`decompose(f,1);` computes a full decomposition, i.e. if f is a composite, then an ideal $(g_1, \ldots, g_m, h)$ is returned, where
g_i are univariate and each entry is primitive such that
$f = g_1 \circ \ldots \circ g_m \circ h$.
If f is not a composite, for instance if $\deg(f)$ is prime, then `decompose(f);` returns f.
The command `decompose` is the inverse: `compose(decompose(f,1))==f`.
Recall, that Chebyshev polynomials of the first kind commute by composition.

The decomposition algorithms work in the tame case, that is if char(basering)=0 or p:=char(basering) > 0 but deg(g) is not divisible by p. Additionally, it works for monic polynomials over Z and in some cases for monic polyomials over coefficient rings.
See `is_composite` for examples. (It also works over the reals but there it seems not be numerical stable.)

More information on the univariate resp. multivariate case.

Univariate decomposition is created, with the additional assumption
$\deg(g), \deg(h) > 1$.

A multivariate polynomial f is a composite, if f can be written as
$g \circ h$, where g is a univariate polynomial and h
is multivariate. Note, that unlike in the univariate case, the polynomial
h may be of degree 1 .
E.g. $f = (x + y)^2 + 2(x + y) + 1$ is the composite of
$g = x^2 + 2x + 1$ and $h = x + y$.

If `nvars(basering)>1`, then, by default, a single-variable multivariate polynomial is not considered to be the same as in the one-variable polynomial ring; it will always be decomposed. That is:

```
> ring r1=0,x,dp;
> decompose(x3+2x+1);
x3+2x+1
```

but:

```
> ring r2=0,(x,y),dp;
> decompose(x3+2x+1);
_[1]=x3+2x+1
_[2]=x
```

In particular:
`is_composite(x3+2x+1)==1;` in `ring r1` but
`is_composite(x3+2x+1)==0;` in `ring r2`.

This is justified by interpreting the polynomial decomposition as an affine Stein factorization of the mapping $f : k^n \to k, n \geq 2$.
The behaviour can changed by the some global variables.
`int DECMETH;` choose von zur Gathen's or Kozen-Landau's method.
`int MINS;` compute f = g o h, such that h(0) = 0.

`int IMPROVE;` simplify the coefficients of g and h if f is not monic.
`int DEGONE;` single-variable multivariate are considered uni-variate.

See `decompopts;` for more information.
Additional information is displayed if `printlevel > 0`.

References:

D. Kozen, S. Landau: Polynomial Decomposition Algorithms,
J. Symb. Comp. (1989), 7, 445-456.
J. von zu Gathen: Functional Decomposition of Polynomials: the Tame Case,
J. Symb. Comp. (1990), 9, 281-299.
J. von zur Gathen, J. Gerhard: Modern computer algebra,
Cambridge University Press, Cambridge, 2003.

Procedures: Auxiliary procedures:

D.4.6.1 decompopts

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: decompopts(); or decompopts("reset");

Return: nothing

Note: in the first case, it shows the setting of the control parameters;
in the second case, it kills the user-defined control parameters and
resets to the default setting which will then be displayed.

int DECMETH; Method for computing the univariate decomposition
0 : (default) Kozen-Landau
1 : von zur Gathen

int IMPROVE Choice of coefficients for the decomposition

$(g_1,\ldots,g_l,h)$ of a non-monic polynomials f.
0 : leadcoef(g_1) = leadcoef(f) and $g_2,\ldots,g_l,h$ are monic
1 : (default), content(g_i) = 1

int MINS

$f=g\circ h,(g_1,\ldots,g_m,h)$ of a non-monic polynomials f.
0 : g(0) = f(0), h(0) = 0 [ueberlegen fuer complete]
1 : (default), g(0)=0, h(0) = f(0)
2 : Tschirnhaus

int DECORD; The order in which the decomposition will be computed
0 : minfirst
1 : (default) maxfirst

int DEGONE; decompose also polynomials built on linear ones
0 : (default)
1 :

Example:

```
LIB "decomp.lib";
decompopts();
↦ 
↦     === Global variables for decomp.lib ===
↦ 
↦   -- DECMETH (int) not defined, implicitly 1
↦   -- MINS (int) not defined, implicitly 0
↦   -- IMPROVE (int) not defined, implicitly 1
```

D.4.6.2 decompose

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: decompose(f); f poly
decompose(f,1); f poly

Return: poly, the input, if f is not a composite
ideal, if the input is a composite

Note: computes a full decomposition if called by the second variant

See: compose

Example:

```
LIB "decomp.lib";
ring r2 = 0,(x,y),dp;
decompose(((x3+2y)^6+x3+2y)^4);
↦ _[1]=x24+4x19+6x14+4x9+x4
↦ _[2]=x3+2y
// complete decomposition
decompose(((x3+2y)^6+x3+2y)^4,1);
↦ _[1]=x2
↦ _[2]=x2
↦ _[3]=x6+x
↦ _[4]=x3+2y
// ----------------------------------------------------------------------
// decompose over the integers
ring rZ = integer,x,dp;
decompose(compose(ideal(x3,x2+2x,x3+2)),1);
↦ _[1]=x3
↦ _[2]=x2-1
↦ _[3]=x3+3
// ----------------------------------------------------------------------
// prime characteristic
ring r7 = 7,x,dp;
decompose(compose(ideal(x2+x,x7)));   // tame case
↦ _[1]=x2+x
↦ _[2]=x7
// ----------------------------------------------------------------------
decompose(compose(ideal(x7+x,x2)));   // wild case
↦ x14+x2
// ----------------------------------------------------------------------
ring ry = (0,y),x,dp;     // y is now a parameter
compose(x2+yx+5,x5-2yx3+x);
```

```
↦ x10+(-4y)*x8+(4y2+2)*x6+(y)*x5+(-4y)*x4+(-2y2)*x3+x2+(y)*x+5
decompose(_);
↦ _[1]=1/4*x2+(-y2+20)/4
↦ _[2]=2*x5+(-4y)*x3+2*x+(y)
// Usage of variable IMPROVE
ideal J = x2+10x, 64x7-112x5+56x3-7x, 4x3-3x;
decompose(compose(J),1);
↦ _[1]=x2+10*x
↦ _[2]=64*x7-112*x5+56*x3-7*x
↦ _[3]=4*x3-3*x
int IMPROVE=0;
exportto(Decomp,IMPROVE);
decompose(compose(J),1);
↦ _[1]=1099511627776*x2+10485760*x
↦ _[2]=x7-7/64*x5+7/2048*x3-7/262144*x
↦ _[3]=x3-3/4*x
```

D.4.6.3 is_composite

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: is_composite(f); f poly

Return: int
1, if f is decomposable
0, if f is not decomposable
-1, if char(basering)>0 and deg(f) is divisible by char(basering) but no decomposition has been found.

Note: The last case means that it could exist a decomposition f=g o h with char(basering)|deg(g), but this wild case cannot be decided by the algorithm.
Some additional information will be displayed when called by the user.

Example:

```
LIB "decomp.lib";
ring r0 = 0,x,dp;
is_composite(x4+5x2+6);    // biquadratic polynomial
↦ 1
is_composite(2x2+x+1);     // prime degree
↦  The degree is prime.
↦ 0
// ---------------------------------------------------------------------
// polynomial ring with several variables
ring R = 0,(x,y),dp;
// ---------------------------------------------------------------------
// single-variable multivariate polynomials
is_composite(2x+1);
↦ 1
is_composite(2x2+x+1);
↦ 1
// ---------------------------------------------------------------------
// prime characteristic
ring r7 = 7,x,dp;
is_composite(compose(ideal(x2+x,x14)));     // is_composite(x14+x7);
↦ 1
```

```
is_composite(compose(ideal(x14+x,x2)));     // is_composite(x14+x2);
↦ // -- Warning: wild case, cannot decide whether the polynomial has a
↦ // -- decomposition goh with deg(g) divisible by char(basering) = 7.
↦ -1
```

D.4.6.4 chebyshev

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: chebyshev(n); n int, n >= 0
chebyshev(n,c); n int, n >= 0, c number, c!=0

Return: poly, the [monic] nth Chebyshev polynomial of the first kind.
The polynomials are defined in the first variable, say x, of the basering.

Note: The (generalized) Chebyshev polynomials of the first kind can be defined by the recursion: $C_0 = c,\ C_1 = x,\ C_n = 2/c \cdot x \cdot C_{n-1} - C_{n-2},\ n \geq 2, c \neq 0$.

These polynomials commute by composition:

$C_m \circ C_n = C_n \circ C_m$.
For c=1, we obtain the standard (non monic) Chebyshev polynomials

T_n which satisfy $T_n(x) = \cos(n \cdot \arccos(x))$.
For c=2 (default), we obtain the monic Chebyshev polynomials P_n

which satisfy the relation $P_n(x + 1/x) = x^n + 1/x^n$.
By default the monic Chebyshev polynomials are returned:

$P_n =$ `chebyshev(n)` and $T_n =$ `chebyshev(n,1)`.
It holds $P_n(x) = 2 \cdot T_n(x/2)$ and more generally

$C_n(c \cdot x) = c \cdot T_n(x)$
That is `subst(chebyshev(n,c),var(1),c*var(1))= c*chebyshev(n,1)`.

If `char(basering) = 2`, then

$C_0 = 1, C_1 = x, C_2 = 1, C_3 = x$, and so on.

Example:

```
LIB "decomp.lib";
ring r = 0,x,lp;
// The monic Chebyshev polynomials
chebyshev(0);
↦ 2
chebyshev(1);
↦ x
chebyshev(2);
↦ x2-2
chebyshev(3);
↦ x3-3x
// These polynomials commute
compose(chebyshev(2),chebyshev(6)) ==
compose(chebyshev(6),chebyshev(2));
↦ 1
// The standard Chebyshev polynomials
chebyshev(0,1);
↦ 1
chebyshev(1,1);
↦ x
```

```
chebyshev(2,1);
↦ 2x2-1
chebyshev(3,1);
↦ 4x3-3x
// ------------------------------------------------------------------------
// The relation for the various Chebyshev polynomials
5*chebyshev(3,1)==subst(chebyshev(3,5),x,5x);
↦ 1
// ------------------------------------------------------------------------
// char 2 case
ring r2 = 2,x,dp;
chebyshev(2);
↦ 1
chebyshev(3);
↦ x
```

D.4.6.5 compose

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: compose(f1,...,fn); f1,...,fn poly
compose(I); I ideal,

Assume: the ideal consists of n=ncols(I) >= 1 entries,
where I[1],...,I[n-1] are univariate in the same variable
but I[n] may be multivariate.

Return: poly, the composition I[1](I[2](...I[n]))

Note: this procedure is the inverse of decompose

See: decompose

Example:

```
LIB "decomp.lib";
ring r = 0,(x,y),dp;
compose(x3+1,x2,y3+x);
↦ y18+6xy15+15x2y12+20x3y9+15x4y6+6x5y3+x6+1
// or the input as one ideal
compose(ideal(x3+1,x2,x3+y));
↦ x18+6x15y+15x12y2+20x9y3+15x6y4+6x3y5+y6+1
```

D.4.6.6 makedistinguished

Procedure from library `decomp.lib` (see Section D.4.6 [decomp_lib], page 1051).

Usage: makedistinguished(f,vvar); f, vvar poly; where vvar is a ring variable

Return: (poly, ideal): the transformed polynomial and an ideal defining the map which reverses the transformation.

Purpose: let vvar = var(1). Then f is transformed by a random linear coordinate change
phi = (var(1), var(2)+c_2*vvar,...,var(n)+c_n*vvar)
such that phi(f) = f o phi becomes distinguished with respect to vvar. That is, the new polynomial contains the monomial vvar^d, where d is the degree of f.
If already f is distinguished w.r.t. vvar, then f is left unchanged and the retransformation is the identity.

Note 1: (this proc correctly works independent of the term ordering.) to apply the reverse transformation, either define a map or use substitute (to be loaded from poly.lib).

Note 2: If p=char(basering) > 0, then there exist polynomials of degree d>=p, e.g. $(p-1)x^p y + xy^p$, that cannot be transformed to a vvar-distinguished polynomial.
In this case, *p random trials will be made and the proc may leave with an ERROR message.

Example:

```
LIB "decomp.lib";
int randval = system("--random");  // store initial value
system("--random",0815);
ring r = 0,(x,y),dp;
poly g;
map phi;
// -----------------------------------------------------------------------
// Example 1:
poly f = 3xy4 + 2xy2 + x5y3 + x + y6;    // degree 8
// make the polynomial y-distinguished
g, phi = makedistinguished(f,y);
g;
↦ x5y3+5x4y4+10x3y5+10x2y6+5xy7+y8+y6+3xy4+3y5+2xy2+2y3+x+y
phi;
↦ phi[1]=x-y
↦ phi[2]=y
// to reverse the transformation apply the map
f == phi(g);
↦ 1
//  -----------------------------------------------------------------------
// Example 2:
// The following polynomial is already x-distinguished
f = x6+y4+xy;
g,phi = makedistinguished(f,x);
g;                          // f is left unchanged
↦ x6+y4+xy
phi;                        // the transformation is the identity.
↦ phi[1]=x
↦ phi[2]=y
system("--random",randval);      // reset random generator
// -----------------------------------------------------------------------
// Example 3:    // polynomials which cannot be transformed
// If p=char(basering)>0, then (p-1)*x^p*y + x*y^p factorizes completely
// in linear factors, since (p-1)*x^p+x equiv 0 on F_p. Hence,
// such polynomials cannot be transformed to a distinguished polynomial.
ring r3 = 3,(x,y),dp;
makedistinguished(2x3y+xy3,y);
↦    ? it could not be transform to a y-distinguished polynomial.
↦    ? leaving decomp.lib::makedistinguished (0)
```

D.4.7 elim_lib

Library: elim.lib

Purpose: Elimination, Saturation and Blowing up

Procedures:

D.4.7.1 blowup0

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: blowup0(J,C [,W]); J,C,W ideals
C = ideal of center of blowup, J = ideal to be blown up, W = ideal of ambient space

Assume: inclusion of ideals : W in J, J in C.
If not, the procedure replaces J by J+W and C by C+J+W

Return: a ring, say B, containing the ideals C,J,W and the ideals
- bR (ideal defining the blown up basering)
- aS (ideal of blown up ambient space)
- eD (ideal of exceptional divisor)
- tT (ideal of total transform)
- sT (ideal of strict transform)
- bM (ideal of the blowup map from basering to B)
such that B/bR is isomorphic to the blowup ring BC.

Purpose: compute the projective blowup of the basering in the center C, the exceptional locus, the total and strict transform of J, and the blowup map.
The projective blowup is a presentation of the blowup ring BC = R[C] = R + t*C + t^2*C^2 + ... (also called Rees ring) of the ideal C in the ring basering R.

Theory: If basering = K[x1,...,xn] and C = <f1,...,fk> then let B = K[x1,...,xn,y1,...,yk] and aS the preimage in B of W under the map B -> K[x1,...,xn,t], xi -> xi, yi -> t*fi. aS is homogeneous in the variables yi and defines a variety Z=V(aS) in A^n x P^(k-1), the ambient space of the blowup of V(W). The projection Z -> A^n is an isomorphism outside the preimage of the center V(C) in A^n and is called the blowup of the center. The preimage of V(C) is called the exceptional set, the preimage of V(J) is called the total transform of V(J). The strict transform is the closure of (total transform minus the exceptional set).
If C = <x1,...,xn> then aS = <yi*xj - yj*xi | i,j=1,...,n> and Z is the blowup of A^n in 0, the exceptional set is P^(k-1).

Note: The procedure creates a new ring with variables y(1..k) and x(1..n) where n=nvars(basering) and k=ncols(C). The ordering is a block ordering where the x-block has the ordering of the basering and the y-block has ordering dp if C is not homogeneous
resp. the weighted ordering wp(b1,...bk) if C is homogeneous with deg(C[i])=bi.

Example:

```
LIB "elim.lib";
ring r  = 0,(x,y),dp;
poly f  = x2+y3;
ideal C = x,y;           //center of blowup
def B1 = blowup0(f,C);
setring B1;
aS;                      //ideal of blown up ambient space
↦ aS[1]=x(1)*y(2)-x(2)*y(1)
tT;                      //ideal of total transform of f
↦ tT[1]=x(1)*y(2)-x(2)*y(1)
↦ tT[2]=x(2)^3+x(1)^2
```

```
sT;                      //ideal of strict transform of f
↦ sT[1]=x(2)*y(2)^2+y(1)^2
eD;                      //ideal of exceptional divisor
↦ eD[1]=x(2)
↦ eD[2]=x(1)
bM;                      //ideal of blowup map r --> B1
↦ bM[1]=x(1)
↦ bM[2]=x(2)
ring R  = 0,(x,y,z),ds;
poly f  = y2+x3+z5;
ideal C = y2,x,z;
ideal W = z-x;
def B2 = blowup0(f,C,W);
setring B2;
B2;                       //weighted ordering
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering ds
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering wp
↦ //                  : names    y(1) y(2) y(3)
↦ //                  : weights     2    1    1
↦ //        block   3 : ordering C
bR;                       //ideal of blown up R
↦ bR[1]=x(1)*y(1)-x(2)^2*y(2)
↦ bR[2]=x(3)*y(1)-x(2)^2*y(3)
↦ bR[3]=-x(1)*y(3)+x(3)*y(2)
aS;                       //ideal of blown up R/W
↦ aS[1]=x(1)*y(1)-x(2)^2*y(2)
↦ aS[2]=x(1)*y(3)-x(3)*y(2)
↦ aS[3]=x(1)-x(3)
↦ aS[4]=x(3)*y(1)-x(2)^2*y(3)
sT;                       //strict transform of f
↦ sT[1]=y(1)+x(3)^2*y(3)+x(3)^4*y(3)
eD;                       //ideal of exceptional divisor
↦ eD[1]=x(1)*y(1)-x(2)^2*y(2)
↦ eD[2]=x(1)*y(3)-x(3)*y(2)
↦ eD[3]=x(1)
↦ eD[4]=x(3)
↦ eD[5]=x(2)^2*y(3)
↦ eD[6]=x(2)^2
//Note that the different affine charts are {y(i)=1}
```

See also: Section D.5.15.1 [blowUp], page 1528; Section D.5.15.2 [blowUp2], page 1529.

D.4.7.2 elimRing

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: elimRing(vars [,w,str]); vars = product of variables to be eliminated (type poly), w = intvec (specifying weights for all variables), str = string either "a" or "b" (default: w=ringweights, str="a")

Return: a list, say L, with R:=L[1] a ring and L[2] an intvec. The ordering in R is an elimination ordering for the variables appearing in vars depending on "a" resp. "b". Let w1

(resp. w2) be the intvec of weights of the variables to be eliminated (resp. not to be eliminated).
The monomial ordering of R has always 2 blocks, the first block corresponds to the (given) variables to be eliminated.
If str = "a" the first block is a(w1,0..0) and the second block is wp(w) resp. ws(w) if the first variable not to be eliminated is local.
If str = "b" the 1st block has ordering wp(w1) and the 2nd block is wp(w2) resp. ws(w2) if the first variable not to be eliminated is local.
If the basering is a quotient ring P/Q, then R is also a quotient ring with Q replaced by a standard basis of Q w.r.t. the new ordering (parameters are not touched).
The intvec L[2] is the intvec of variable weights (or the given w) with weights <= 0 replaced by 1.

Purpose: Prepare a ring for eliminating vars from an ideal/module by computing a standard basis in R with a fast monomial ordering. This procedure is used by the procedure elim.

Example:

```
LIB "elim.lib";
ring R = 0,(x,y,z,u,v),(c,lp);
def P = elimRing(yu);  P;
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering a
↦ //                  : names    y u x z v
↦ //                  : weights  1 1 0 0 0
↦ //        block   2 : ordering dp
↦ //                  : names    y u x z v
↦ //        block   3 : ordering C
↦ [2]:
↦    1,1,1,1,1
intvec w = 1,1,3,4,5;
elimRing(yu,w);
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering a
↦ //                  : names    y u x z v
↦ //                  : weights  1 4 0 0 0
↦ //        block   2 : ordering wp
↦ //                  : names    y u x z v
↦ //                  : weights  1 1 3 4 5
↦ //        block   3 : ordering C
↦ [2]:
↦    1,1,3,4,5
ring S =  (0,a),(x,y,z,u,v),ws(1,2,3,4,5);
minpoly = a2+1;
qring T = std(ideal(x+y2+v3,(x+v)^2));
def Q = elimRing(yv)[1];
setring Q; Q;
↦ // coefficients: QQ[a]/(a2+1)
↦ // number of vars : 5
↦ //        block   1 : ordering a
↦ //                  : names    y v x z u
↦ //                  : weights  2 5 0 0 0
↦ //        block   2 : ordering ws
↦ //                  : names    y v x z u
↦ //                  : weights  1 2 3 4 5
↦ //        block   3 : ordering C
↦ // quotient ring from ideal
↦ _[1]=y2+2*yu+u2
↦ _[2]=v2+y+u3
```

D.4.7.3 elim

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: elim(id,arg[,s]); id ideal/module, arg can be either an intvec v or a product p of variables (type poly), s a string determining the method which can be "slimgb" or "std" or, additionally, "withWeigts".

Return: ideal/module obtained from id by eliminating either the variables with indices appearing in v or the variables appearing in p. Does not work in a qring.

Method: elim uses elimRing to create a ring with an elimination ordering for the variables to be eliminated and then applies std if "std" is given, or slimgb if "slimgb" is given, or a heuristically chosen method.
If the variables in the basering have weights these weights are used in elimRing. If a string "withWeigts" as (optional) argument is given Singular computes weights for the variables to make the input as homogeneous as possible.
The method is different from that used by eliminate and elim1; depending on the example, any of these commands can be faster.

Note: No special monomial ordering is required, i.e. the ordering can be local or mixed. The result is a SB with respect to the ordering of the second block used by elimRing. E.g. if the first var not to be eliminated is global, resp. local, this ordering is dp, resp. ds (or wp, resp. ws, with the given weights for these variables). If printlevel > 0 the ring for which the output is a SB is shown.

Example:

```
LIB "elim.lib";
ring r=0,(x,y,u,v,w),dp;
ideal i=x-u,y-u2,w-u3,v-x+y3;
elim(i,3..4);
↦ _[1]=y2-xw
↦ _[2]=xy-w
↦ _[3]=x2-y
elim(i,uv);
↦ _[1]=y2-xw
↦ _[2]=xy-w
↦ _[3]=x2-y
int p = printlevel;
printlevel = 2;
elim(i,uv,"withWeights","slimgb");
↦ // result is a SB in the following ring:
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering a
↦ //                  : names    u v x y w
↦ //                  : weights  5 21 0 0 0
↦ //        block   2 : ordering wp
↦ //                  : names    u v x y w
↦ //                  : weights  5 7 5 21 15
↦ //        block   3 : ordering C
↦ _[1]=x3-w
↦ _[2]=-x2+y
printlevel = p;
ring S =  (0,a),(x,y,z,u,v),ws(1,2,3,4,5);
minpoly = a2+1;
ideal i=x-u,y-u2,az-u3,v-x+ay3;
module m=i*gen(1)+i*gen(2);
m=elim(m,xy);
show(m);
↦ // module, 4 generator(s)
↦ [0,z+(a)*u3]
↦ [z+(a)*u3]
↦ [0,u-v+(-a)*u6]
↦ [u-v+(-a)*u6]
```

See also: Section D.4.7.4 [elim1], page 1063; Section 5.1.28 [eliminate], page 176.

D.4.7.4 elim1

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: elim1(id,arg); id ideal/module, arg can be either an intvec v or a product p of variables (type poly)

Return: ideal/module obtained from id by eliminating either the variables with indices appearing in v or the variables appearing in p

Method: elim1 calls eliminate but in a ring with ordering dp (resp. ls) if the first var not to be eliminated belongs to a -p (resp. -s) ordering.

Note: no special monomial ordering is required.

Example:

```
LIB "elim.lib";
ring r=0,(x,y,t,s,z),dp;
ideal i=x-t,y-t2,z-t3,s-x+y3;
elim1(i,ts);
↦ _[1]=y2-xz
↦ _[2]=xy-z
↦ _[3]=x2-y
module m=i*gen(1)+i*gen(2);
m=elim1(m,3..4); show(m);
↦ // module, 6 generator(s)
↦ [y2-xz]
↦ [0,y2-xz]
↦ [xy-z]
↦ [0,xy-z]
↦ [x2-y]
↦ [0,x2-y]
```

See also: Section D.4.7.3 [elim], page 1062; Section 5.1.28 [eliminate], page 176.

D.4.7.5 elim2

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: elim2(id,v); id ideal/module, v intvec

Returns: ideal/module obtained from id by eliminating variables in v

Note: no special monomial ordering is required, result is a SB with respect to ordering dp (resp. ls) if the first var not to be eliminated belongs to a -p (resp. -s) blockordering

Example:

```
LIB "elim.lib";
ring r=0,(x,y,u,v,w),dp;
ideal i=x-u,y-u2,w-u3,v-x+y3;
elim2(i,3..4);
↦ _[1]=y2-xw
↦ _[2]=xy-w
↦ _[3]=x2-y
module m=i*gen(1)+i*gen(2);
m=elim2(m,3..4);show(m);
↦ // module, 6 generator(s)
↦ [y2-xw]
↦ [0,y2-xw]
↦ [xy-w]
↦ [0,xy-w]
↦ [x2-y]
↦ [0,x2-y]
```

See also: Section D.4.7.3 [elim], page 1062; Section D.4.7.4 [elim1], page 1063; Section 5.1.28 [eliminate], page 176.

D.4.7.6 nselect

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: nselect(id,v); id = ideal, module or matrix, v = intvec

Return: generators (or columns) of id not containing the variables with index an entry of v

Example:

```
LIB "elim.lib";
ring r=0,(x,y,t,s,z),(c,dp);
ideal i=x-y,y-z2,z-t3,s-x+y3;
nselect(i,3);
↦ _[1]=x-y
↦ _[2]=-z2+y
↦ _[3]=y3-x+s
module m=i*(gen(1)+gen(2));
m;
↦ m[1]=[x-y,x-y]
↦ m[2]=[-z2+y,-z2+y]
↦ m[3]=[-t3+z,-t3+z]
↦ m[4]=[y3-x+s,y3-x+s]
nselect(m,3..4);
↦ _[1]=[x-y,x-y]
↦ _[2]=[-z2+y,-z2+y]
```

```
nselect(matrix(m),3..4);
↦ _[1,1]=x-y
↦ _[1,2]=-z2+y
↦ _[2,1]=x-y
↦ _[2,2]=-z2+y
```

See also: Section D.4.7.9 [select], page 1066; Section D.4.7.10 [select1], page 1066.

D.4.7.7 sat

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: sat(id,j); id=ideal/module, j=ideal

Return: ideal/module
saturation of id with respect to j (= union_(k=1...) of id:j^k)

Note: result is a standard basis in the basering

Example:

```
LIB "elim.lib";
ring r     = 2,(x,y,z),dp;
poly F     = x5+y5+(x-y)^2*xyz;
ideal j    = jacob(F);
sat(j,maxideal(1));
↦ _[1]=x3+x2y+xy2+y3
↦ _[2]=y4+x2yz+y3z
↦ _[3]=x2y2+y4
sat(j,maxideal(2));
↦ _[1]=x3+x2y+xy2+y3
↦ _[2]=y4+x2yz+y3z
↦ _[3]=x2y2+y4
```

See also: Section D.4.14.2 [modSat], page 1121; Section D.4.7.8 [sat_with_exp], page 1065.

D.4.7.8 sat_with_exp

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: sat(id,j); id=ideal/module, j=ideal

Return: list of an ideal/module [1] and an integer [2]:
[1] = saturation of id with respect to j (= union_(k=1...) of id:j^k) [2] = saturation exponent (= min(k | id:j^k = id:j^(k+1)))

Note: [1] is a standard basis in the basering

Example:

```
LIB "elim.lib";
ring r     = 2,(x,y,z),dp;
poly F     = x5+y5+(x-y)^2*xyz;
ideal j    = jacob(F);
sat(j,maxideal(1));
↦ _[1]=x3+x2y+xy2+y3
↦ _[2]=y4+x2yz+y3z
↦ _[3]=x2y2+y4
sat(j,maxideal(2));
↦ _[1]=x3+x2y+xy2+y3
↦ _[2]=y4+x2yz+y3z
↦ _[3]=x2y2+y4
```

See also: Section D.4.14.2 [modSat], page 1121; Section D.4.7.7 [sat], page 1065.

D.4.7.9 select

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: select(id,n[,m]); id = ideal/module/matrix, v = intvec

Return: generators/columns of id containing all variables with index an entry of v

Note: use 'select1' for selecting generators/columns containing at least one of the variables with index an entry of v

Example:

```
LIB "elim.lib";
ring r=0,(x,y,t,s,z),(c,dp);
ideal i=x-y,y-z2,z-t3,s-x+y3;
ideal j=select(i,1);
j;
↦ j[1]=x-y
↦ j[2]=y3-x+s
module m=i*(gen(1)+gen(2));
m;
↦ m[1]=[x-y,x-y]
↦ m[2]=[-z2+y,-z2+y]
↦ m[3]=[-t3+z,-t3+z]
↦ m[4]=[y3-x+s,y3-x+s]
select(m,1..2);
↦ _[1]=[x-y,x-y]
↦ _[2]=[y3-x+s,y3-x+s]
select(matrix(m),1..2);
↦ _[1,1]=x-y
↦ _[1,2]=y3-x+s
↦ _[2,1]=x-y
↦ _[2,2]=y3-x+s
```

See also: Section D.4.7.6 [nselect], page 1064; Section D.4.7.10 [select1], page 1066.

D.4.7.10 select1

Procedure from library `elim.lib` (see Section D.4.7 [elim_lib], page 1058).

Usage: select1(id,v); id = ideal/module/matrix, v = intvec

Return: generators/columns of id containing at least one of the variables with index an entry of v

Note: use 'select' for selecting generators/columns containing all variables with index an entry of v

Example:

```
LIB "elim.lib";
ring r=0,(x,y,t,s,z),(c,dp);
ideal i=x-y,y-z2,z-t3,s-x+y3;
ideal j=select1(i,1);j;
```

```
↦ j[1]=x-y
↦ j[2]=y3-x+s
module m=i*(gen(1)+gen(2)); m;
↦ m[1]=[x-y,x-y]
↦ m[2]=[-z2+y,-z2+y]
↦ m[3]=[-t3+z,-t3+z]
↦ m[4]=[y3-x+s,y3-x+s]
select1(m,1..2);
↦ _[1]=[x-y,x-y]
↦ _[2]=[-z2+y,-z2+y]
↦ _[3]=[y3-x+s,y3-x+s]
select1(matrix(m),1..2);
↦ _[1,1]=x-y
↦ _[1,2]=-z2+y
↦ _[1,3]=y3-x+s
↦ _[2,1]=x-y
↦ _[2,2]=-z2+y
↦ _[2,3]=y3-x+s
```

See also: Section D.4.7.6 [nselect], page 1064; Section D.4.7.9 [select], page 1066.

D.4.8 ellipticcovers_lib

Library: ellipticCovers.lib

Purpose: Gromov-Witten numbers of elliptic curves

Authors: J. Boehm, boehm @ mathematik.uni-kl.de
A. Buchholz, buchholz @ math.uni-sb.de
H. Markwig hannah @ math.uni-sb.de

Overview: We implement a formula for computing the number of covers of elliptic curves. It has beed obtained by proving mirror symmetry
for arbitrary genus by tropical methods in [BBM]. A Feynman graph of genus g is a trivalent, connected graph of genus g (with 2g-2 vertices and 3g-3 edges). The branch type b=(b_1,...,b_(3g-3)) of a stable map is the multiplicity of the the edge i over a fixed base point.

Given a Feynman graph G and a branch type b, we obtain the number N_(G,b) of stable maps of branch type b from a genus g curve of topological type G to the elliptic curve by computing a path integral
over a rational function. The path integral is computed as a residue.

The sum of N_(G,b) over all branch types b of sum d gives N_(G,d)*|Aut(G)|, with the Gromov-Witten invariant N_(G,d) of degree d stable maps from a genus g curve of topological type G to the elliptic curve.

The sum of N_(G,d) over all such graphs gives the usual Gromov-Witten invariant N_(g,d) of degree d stable maps from a genus g curve to the elliptic curve.

The key function computing the numbers N_(G,b) and N_(G,d) is gromovWitten.

References:

[BBM] J. Boehm, A. Buchholz, H. Markwig: Tropical mirror symmetry for elliptic curves, arXiv:1309.5893 (2013).

Types: graph

Procedures:

D.4.8.1 makeGraph

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: makeGraph(v,e); v list, e list

Assume: v is a list of integers, e is a list of two element lists of v.

Return: graph with vertices v and edges e

Theory: Creates a graph from a list of vertices and edges. The vertices can be any type.

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
G;
↦ [[1, 3], [1, 2], [1, 2], [2, 4], [3, 4], [3, 4]]
↦ Graph with 4 vertices and 6 edges
↦
```

D.4.8.2 printGraph

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: printGraph(G); G graph

Assume: G is a graph.

Theory: This is the print function used by Singular to print a graph.

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
G;
↦ [[1, 3], [1, 2], [1, 2], [2, 4], [3, 4], [3, 4]]
↦ Graph with 4 vertices and 6 edges
↦
```

D.4.8.3 propagator

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: propagator(xy,d); xy list, d int
propagator(G,b); G graph, b list

Assume: xy is a list of two numbers x and y in a rational function field, d non-negative integer.
G is a Feynman graph, a is a list of integers of length equal to the number of edges of G.
We assume that the coefficient ring has one rational variable for each vertex of G.

Return: number, the propagator associated to the input data.

Theory: If xy and d are specified, then the function returns x^2*y^2/(x^2-y^2)^2) for d=0, which is a associated to an edge with vertices x and y not passing above the base point. For d>0 it returns the sum of (j*x^(4*j)+j*y^(4*j))/(x*y)^(2*j) over all divisors j of d, which is associated to an edge with vertices x and y passing with multiplicity d above the base point.

Essentially the variables x and y stand for the position of the base points.

In the second way of using this function, G is a Feynman graph and b is a branch type over a fixed base point of a cover with source G and target an elliptic curve. It returns the product of propagator(list(v[i],w[i]),b[i]) over all edges i with multiplicity b[i] over the base point and vertices v[i] and w[i].

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
propagator(list(x1,x2),0);
↦ (x1^2*x2^2)/(x1^4-2*x1^2*x2^2+x2^4)
propagator(list(x1,x2),2);
↦ (2*x1^8+x1^6*x2^2+x1^2*x2^6+2*x2^8)/(x1^4*x2^4)
propagator(G,list(1,1,1,0,0,0));
↦ (x1^12*x3^2*x4^6+2*x1^8*x2^4*x3^2*x4^6+x1^8*x3^6*x4^6+x1^4*x2^8*x3^2*x4^6\
   +2*x1^4*x2^4*x3^6*x4^6+x2^8*x3^6*x4^6)/(x1^6*x2^6*x3^8-4*x1^6*x2^6*x3^6*x\
   4^2+6*x1^6*x2^6*x3^4*x4^4-4*x1^6*x2^6*x3^2*x4^6+x1^6*x2^6*x4^8-2*x1^6*x2^\
   4*x3^8*x4^2+8*x1^6*x2^4*x3^6*x4^4-12*x1^6*x2^4*x3^4*x4^6+8*x1^6*x2^4*x3^2\
   *x4^8-2*x1^6*x2^4*x4^10+x1^6*x2^2*x3^8*x4^4-4*x1^6*x2^2*x3^6*x4^6+6*x1^6*\
   x2^2*x3^4*x4^8-4*x1^6*x2^2*x3^2*x4^10+x1^6*x2^2*x4^12)
```

D.4.8.4 computeConstant

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: computeConstant(f,x); f number, x number

Assume: f is a number in a rational function field, x is a variable of the field.

Return: number, the constant coefficient of the Laurent series of f in the variable x.

Theory: Computes the constant coefficient of the Laurent series by iterative differentiation.

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
number P = propagator(G,list(1,1,1,0,0,0));
computeConstant(P,x2);
↦ (2*x1^6*x3^2+2*x1^2*x3^6)/(x3^8-4*x3^6*x4^2+6*x3^4*x4^4-4*x3^2*x4^6+x4^8)
```

D.4.8.5 evalutateIntegral

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

D.4.8.6 gromovWitten

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: gromovWitten(P); P number
gromovWitten(G,d); G graph, d int
gromovWitten(G,b); G graph, b list

Assume: P is a propagator, or
G is a Feynman graph and d a non-negative integer, or
G is a Feynman graph and b is a list of integers of length equal to the number of edges of G
We assume that the coefficient ring has one rational variable for each vertex of G.

Return: Gromov-Witten invariant.

Theory: Computes

- the Gromov-Witten invariant of a given propagator P, or

- the invariant N_(G,d)*|Aut(G)| where d is the degree of the covering, or

- the number N_(G,b) of coverings with source G and target an elliptic curves with branch type a over a fixed base point (that is, the i-th edge passes over the base point with multiplicity b[i]).

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
number P = propagator(G,list(0,2,1,0,0,1));
gromovWitten(P);
↦ 256
gromovWitten(G,list(0,2,1,0,0,1));
↦ 256
gromovWitten(G,2);
↦ 32
```

D.4.8.7 computeGromovWitten

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: computeGromovWitten(G, d, st, en [, vb]); G graph, d int, st int, en int, optional: vb int

Assume: G is a Feynman graph, d a non-negative integer, st specified the start- and en the end partition in the list pa = partition(d). Specifying a positive optional integer vb leads to intermediate printout.
We assume that the coefficient ring has one rational variable for each vertex of G.

Return: list L, where L[i] is gromovWitten(G,pa[i]) and all others are zero.

Theory: This function does essentially the same as the function gromovWitten, but is designed for handling complicated examples. Eventually it will also run in parallel.

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q1,q2,q3,q4,q5,q6),dp;
graph G = makeGraph(list(1,2,3,4),list(list(1,3),list(1,2),list(1,2),list(2,4),list(3
partitions(6,2);
↦ [1]:
↦    0,0,0,0,0,2
↦ [2]:
↦    0,0,0,0,1,1
↦ [3]:
↦    0,0,0,0,2,0
↦ [4]:
↦    0,0,0,1,0,1
↦ [5]:
↦    0,0,0,1,1,0
↦ [6]:
↦    0,0,0,2,0,0
↦ [7]:
↦    0,0,1,0,0,1
↦ [8]:
↦    0,0,1,0,1,0
↦ [9]:
↦    0,0,1,1,0,0
↦ [10]:
↦    0,0,2,0,0,0
↦ [11]:
↦    0,1,0,0,0,1
↦ [12]:
↦    0,1,0,0,1,0
↦ [13]:
↦    0,1,0,1,0,0
↦ [14]:
↦    0,1,1,0,0,0
↦ [15]:
↦    0,2,0,0,0,0
↦ [16]:
↦    1,0,0,0,0,1
↦ [17]:
↦    1,0,0,0,1,0
↦ [18]:
↦    1,0,0,1,0,0
↦ [19]:
↦    1,0,1,0,0,0
↦ [20]:
↦    1,1,0,0,0,0
↦ [21]:
↦    2,0,0,0,0,0
computeGromovWitten(G,2,3,7);
```

```
↦ [1]:
↦ 0
↦ [2]:
↦ 0
↦ [3]:
↦ 0
↦ [4]:
↦ 0
↦ [5]:
↦ 0
↦ [6]:
↦ 8
↦ [7]:
↦ 0
↦ [8]:
↦ 0
↦ [9]:
↦ 0
↦ [10]:
↦ 0
↦ [11]:
↦ 0
↦ [12]:
↦ 0
↦ [13]:
↦ 0
↦ [14]:
↦ 0
↦ [15]:
↦ 0
↦ [16]:
↦ 0
↦ [17]:
↦ 0
↦ [18]:
↦ 0
↦ [19]:
↦ 0
↦ [20]:
↦ 0
↦ [21]:
↦ 0
computeGromovWitten(G,2,3,7,1);
↦ 21
↦ 3 / 21    0,0,0,0,2,0     0      0     0
↦ 4 / 21    0,0,0,1,0,1     0      0     0
↦ 5 / 21    0,0,0,1,1,0     0      0     0
↦ 6 / 21    0,0,0,2,0,0     8      8     0
↦ 7 / 21    0,0,1,0,0,1     0      8     0
↦ [1]:
↦ 0
↦ [2]:
↦ 0
↦ [3]:
↦ 0
↦ [4]:
↦ 0
↦ [5]:
↦ 0
↦ [6]:
↦ 8
↦ [7]:
↦ 0
↦ [8]:
↦ 0
↦ [9]:
↦ 0
↦ [10]:
↦ 0
↦ [11]:
↦ 0
↦ [12]:
↦ 0
↦ [13]:
↦ 0
↦ [14]:
↦ 0
↦ [15]:
↦ 0
↦ [16]:
↦ 0
↦ [17]:
↦ 0
↦ [18]:
↦ 0
↦ [19]:
↦ 0
↦ [20]:
↦ 0
↦ [21]:
↦ 0
```

D.4.8.8 partitions

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: partitions(n,a); n int, a int

Assume: n and a are positive integers

Return: list of all partitions of a into n summands.

Theory: Computes all partitions of a into n summands.
This may eventually be deleted and become a more efficient kernel function.

Example:

```
LIB "ellipticcovers.lib";
partitions(3,7);
↦ [1]:
↦    0,0,7
↦ [2]:
↦    0,1,6
↦ [3]:
↦    0,2,5
↦ [4]:
↦    0,3,4
↦ [5]:
↦    0,4,3
↦ [6]:
↦    0,5,2
↦ [7]:
↦    0,6,1
↦ [8]:
↦    0,7,0
↦ [9]:
↦    1,0,6
↦ [10]:
↦    1,1,5
↦ [11]:
↦    1,2,4
↦ [12]:
↦    1,3,3
↦ [13]:
↦    1,4,2
↦ [14]:
↦    1,5,1
↦ [15]:
↦    1,6,0
↦ [16]:
↦    2,0,5
↦ [17]:
↦    2,1,4
↦ [18]:
↦    2,2,3
↦ [19]:
↦    2,3,2
↦ [20]:
↦    2,4,1
↦ [21]:
↦    2,5,0
↦ [22]:
↦    3,0,4
↦ [23]:
↦    3,1,3
↦ [24]:
↦    3,2,2
↦ [25]:
↦    3,3,1
↦ [26]:
```

```
↦    3,4,0
↦ [27]:
↦    4,0,3
↦ [28]:
↦    4,1,2
↦ [29]:
↦    4,2,1
↦ [30]:
↦    4,3,0
↦ [31]:
↦    5,0,2
↦ [32]:
↦    5,1,1
↦ [33]:
↦    5,2,0
↦ [34]:
↦    6,0,1
↦ [35]:
↦    6,1,0
↦ [36]:
↦    7,0,0
```

D.4.8.9 permute

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: permute(N); N list

Assume: N is a list

Return: list with all permutations of N.

Theory: Computes all permutations of N.
This will eventually be deleted and become a more efficient kernel function.

Example:

```
LIB "ellipticcovers.lib";
ring R=(0,x1,x2,x3,x4),(q),dp;
permute(list(x1,x2,x3,x4));
↦ [1]:
↦    [1]:
↦ (x4)
↦    [2]:
↦ (x3)
↦    [3]:
↦ (x2)
↦    [4]:
↦ (x1)
↦ [2]:
↦    [1]:
↦ (x3)
↦    [2]:
↦ (x4)
↦      [3]:
↦   (x2)
↦      [4]:
↦   (x1)
↦   [3]:
↦      [1]:
↦   (x4)
↦      [2]:
↦   (x2)
↦      [3]:
↦   (x3)
↦      [4]:
↦   (x1)
↦   [4]:
↦      [1]:
↦   (x2)
↦      [2]:
↦   (x4)
↦      [3]:
↦   (x3)
↦      [4]:
↦   (x1)
↦   [5]:
↦      [1]:
↦   (x3)
↦      [2]:
↦   (x2)
↦      [3]:
↦   (x4)
↦      [4]:
↦   (x1)
↦   [6]:
↦      [1]:
↦   (x2)
↦      [2]:
↦   (x3)
↦      [3]:
↦   (x4)
↦      [4]:
↦   (x1)
↦   [7]:
↦      [1]:
↦   (x4)
↦      [2]:
↦   (x3)
↦      [3]:
↦   (x1)
↦      [4]:
↦   (x2)
↦   [8]:
↦      [1]:
↦   (x3)
↦      [2]:
```

```
↦ (x4)
↦    [3]:
↦ (x1)
↦    [4]:
↦ (x2)
↦ [9]:
↦    [1]:
↦ (x4)
↦    [2]:
↦ (x1)
↦    [3]:
↦ (x3)
↦    [4]:
↦ (x2)
↦ [10]:
↦    [1]:
↦ (x1)
↦    [2]:
↦ (x4)
↦    [3]:
↦ (x3)
↦    [4]:
↦ (x2)
↦ [11]:
↦    [1]:
↦ (x3)
↦    [2]:
↦ (x1)
↦    [3]:
↦ (x4)
↦    [4]:
↦ (x2)
↦ [12]:
↦    [1]:
↦ (x1)
↦    [2]:
↦ (x3)
↦    [3]:
↦ (x4)
↦    [4]:
↦ (x2)
↦ [13]:
↦    [1]:
↦ (x4)
↦    [2]:
↦ (x2)
↦    [3]:
↦ (x1)
↦    [4]:
↦ (x3)
↦ [14]:
↦    [1]:
↦ (x2)
↦     [2]:
↦  (x4)
↦     [3]:
↦  (x1)
↦     [4]:
↦  (x3)
↦  [15]:
↦     [1]:
↦  (x4)
↦     [2]:
↦  (x1)
↦     [3]:
↦  (x2)
↦     [4]:
↦  (x3)
↦  [16]:
↦     [1]:
↦  (x1)
↦     [2]:
↦  (x4)
↦     [3]:
↦  (x2)
↦     [4]:
↦  (x3)
↦  [17]:
↦     [1]:
↦  (x2)
↦     [2]:
↦  (x1)
↦     [3]:
↦  (x4)
↦     [4]:
↦  (x3)
↦  [18]:
↦     [1]:
↦  (x1)
↦     [2]:
↦  (x2)
↦     [3]:
↦  (x4)
↦     [4]:
↦  (x3)
↦  [19]:
↦     [1]:
↦  (x3)
↦     [2]:
↦  (x2)
↦     [3]:
↦  (x1)
↦     [4]:
↦  (x4)
↦  [20]:
↦     [1]:
```

```
↦ (x2)
↦    [2]:
↦ (x3)
↦    [3]:
↦ (x1)
↦    [4]:
↦ (x4)
↦ [21]:
↦    [1]:
↦ (x3)
↦    [2]:
↦ (x1)
↦    [3]:
↦ (x2)
↦    [4]:
↦ (x4)
↦ [22]:
↦    [1]:
↦ (x1)
↦    [2]:
↦ (x3)
↦    [3]:
↦ (x2)
↦    [4]:
↦ (x4)
↦ [23]:
↦    [1]:
↦ (x2)
↦    [2]:
↦ (x1)
↦    [3]:
↦ (x3)
↦    [4]:
↦ (x4)
↦ [24]:
↦    [1]:
↦ (x1)
↦    [2]:
↦ (x2)
↦    [3]:
↦ (x3)
↦    [4]:
↦ (x4)
```

D.4.8.10 lsum

Procedure from library `ellipticcovers.lib` (see Section D.4.8 [ellipticcovers_lib], page 1067).

Usage: lsum(L); L list

Assume: L is a list of things with the binary operator `+` defined.

Return: The sum of the elements of L.

Theory: Sums the elements of a list.

Eventually this will be deleted and become a more efficient kernel function.

Example:

```
LIB "ellipticcovers.lib";
list L = 1,2,3,4,5;
lsum(L);
↦ 15
```

D.4.9 ffmodstd_lib

Library: ffmodstd.lib

Purpose: Groebner bases of ideals in polynomial rings over rational function fields

Authors: D.K. Boku boku@mathematik.uni-kl.de
W. Decker decker@mathematik.uni-kl.dei
C. Fieker fieker@mathematik.uni-kl.de
A. Steenpass steenpass@mathematik.uni-kl.de

Overview: A library for computing a Groebner basis of an ideal in a polynomial ring over an algebraic function field Q(T):=Q(t_1,...,t_m) using modular methods and sparse multivariate rational interpolation, where the t_i are transcendental over Q. The idea is as follows: Given an ideal I in Q(T)[X], we map I to J via the map sending T to Tz:=(t_1z+s_1,..., t_mz+s_m) for a suitable point s in Q^m\{(0,...,0)} and for some extra variable z so that J is an ideal in Q(Tz)[X]. For a suitable point b in Z^m\{(0,...,0)}, we map J to K via the map sending (T,z) to (b,z), where b:=(b_1,...,b_m) (usually the b_i's are distinct primes), so that K is an ideal in Q(z)[X]. For such a rational point b, we compute a Groebner basis G_b of K using modular algorithms [1], where prime numbers are replaced by maximal ideals of the form <z-z_i>, and univariate rational interpolation [2,7]. Note that since Q[z]/<z-z_i> = Q we also use (if required) modular algorithms [1] over Q. The procedure is repeated for many rational points b until their number is sufficiently large to recover the correct coefficients in Q(T). Once we have these points, we obtain a set of polynomials G by applying the sparse multivariate rational interpolation algorithm from [4] coefficient- wise to the list of Groebner bases G_b in Q(z)[X], where this algorithm makes use of the following algorithms: univariate polynomial interpolation [2], univariate rational function reconstruction [7], and multivariate polynomial interpolation [3]. The last algorithm uses the well-known Berlekamp/Massey algorithm [5] and its early termination version [6]. The set G is then a Groebner basis of I with high probability.

References:

[1] E. A. Arnold: Modular algorithms for computing Groebner bases. J. Symb. Comput. 35, 403-419 (2003).
[2] R. L. Burden and J. D. Faires: Numerical analysis. 9th ed. (1993). [3] M. Ben-Or and P. Tiwari: A deterministic algorithm for sparse multivariate polynomial interpolation. Proc. of the 20th Annual ACM Symposium on Theory of Computing, 301-309 (1988).
[4] A. Cuyt and W.-s. Lee: Sparse interpolation of multivariate rational functions. Theor. Comput. Sci. 412, 1445-1456 (2011).
[5] E. Kaltofen and W.-s. Lee: Early termination in sparse interpolation algorithms. J. Symb. Comput. 36, 365-400 (2003).

[6] E. Kaltofen, W.-s. Lee and A. A. Lobo: Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. Proc. ISSAC (ISSAC '00), 192-201 (2000).
[7] K. Sara and M. Monagan: Fast Rational Function Reconstruction. Proc. ISSAC (ISSAC '06), 184-190 (2006).

Procedures:

D.4.9.1 fareypoly

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: fareypoly(f, g[, m]); f poly, g poly, m int

Return: a list l where r/t (r:=l[1], t:=l[2]) is a univariate rational function such that r/t = g mod f, gcd(r,t)=gcd(f,t)=1 and deg(r) + deg(t) < deg(f)

Note: An optional parameter m can be provided to define the way how t is normalized. If m = 0 (default), then the leading coefficient of t is 1. Otherwise, assuming the polynomial t has a non-zero constant term, the procedure returns the uniquely determined rational function r/t where the constant term in t is equal to 1.
If the ground ring has n variables and f and g are in a polynomial ring k[var(i)] (k is a field) for some i<=n, then the function r/t is returned as an element in k(var(i)).
In positive characteristic, the condition r/t = g mod f may not be satisfied. The degree deg(f) of f must be higher than the degree deg(g) of g.

Example:

```
LIB "ffmodstd.lib";
ring rr=23,x,dp;
poly g = 10x5-5x4+3x3+3x2-x-11;
poly f = x6+2x5-9x4+x3-9x2+7x+7;
fareypoly(g,f);
↦ [1]:
↦    -11x3-7x2-3x+6
↦ [2]:
↦    x-9
fareypoly(g,f,1);
↦ [1]:
↦    -9x3+11x2+8x+7
↦ [2]:
↦    5x+1
ring R = 0, x,dp;
poly g = (24/1616615)*x6-(732/1616615)*x5+(9558/1616615)*x4-(14187/323323)*x3+
(1148101/1616615)*x2+(4089347/1616615)*x+547356/230945;
poly f = x7-28x6+322x5-1960x4+6769x3-13132x2+13068x-5040;
fareypoly(g,f);
↦ [1]:
↦    1/2x3+9/2x2+17/2x+6
↦ [2]:
↦    x+5/2
fareypoly(g,f,1);
↦ [1]:
↦    1/5x3+9/5x2+17/5x+12/5
↦ [2]:
↦    2/5x+1
```

```
ring r = (499,a),x,dp;
number N = (-113a4+170a3-29a2+226a+222)/(a7-56a6+114a5+144a4+171a3-64a2+192a);
poly h1 = x4+(-55a5-18a4-141a3+233a2+66a-40)/(a4-28a3+40a2-2a+210)*x3;
poly h2 = (107a6-221a5-68a4-93a3+112a2-54a+216)/(a4-28a3+40a2-2a+210)*x2;
poly h3 = (-53a7+214a6+27a5+12a4+15a3+60a2-167a-83)/(a4-28a3+40a2-2a+210)*x;
poly h4 = (10a6-75a5+47a4+246a3-20a2-217a+196)/(a4-28a3+40a2-2a+210);
poly g = N*(h1+h2+h3+h4);
poly f = x5+(-2a-119)*x4+(a2+237a+3437)*x3+(-118a2-6756a-29401)*x2+
(3319a2+55483a+26082)*x+(-26082a2-26082a);
fareypoly(g,f);
↦ [1]:
↦    (a)*x+2
↦ [2]:
↦    x2-3*x+(a)
```

See also: Section 5.1.37 [farey], page 181; Section D.4.9.2 [polyInterpolation], page 1082.

D.4.9.2 polyInterpolation

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: polyInterpolation(d, e[, n, L]); d list, e list, n int, L list

Return: a list l_p where f:=l_p[1] is a polynomial of degree at most size(d)-1 which satisfies the conditions f(d[i])=e[i] for all i, l_p[2] is the product of all (var(n)-d[i]) for 1 <= i <= size(d) and l_p[3]=d.

Note: The procedure applies the Newton interpolation algorithm to the pair (d,e) and returns the output w.r.t. the first variable (default) of the ground ring. If an optional parameter n, 1<=n<=N (N is the number of variables in the current basering), is given, then the procedure returns the list l_p w.r.t. the n-th variable. Moreover, if the number of points (d'[i],e'[i]) is not large enough to obtain the target polynomial, L = polyInterpolation(d', e', n) can be provided as an optional parameter to add more interpolation points. The elements in the first list must be distinct.

Example:

```
LIB "ffmodstd.lib";
ring rr = 23,(x,y),dp;
list d = 1,2,3,4;
list e = -1,10,3,8;
polyInterpolation(d,e);
↦ [1]:
↦    5x3+7x2+x+9
↦ [2]:
↦    x4-10x3-11x2-4x+1
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦    [3]:
↦       3
↦    [4]:
↦       4
polyInterpolation(d,e,2)[1];
```

```
↦ 5y3+7y2+y+9
list d1 = 5,6;
list e1 = -7,6;
list L = polyInterpolation(d,e);
L = polyInterpolation(d1,e1,1,L); // add points
L;
↦ [1]:
↦    10x5-5x4+3x3+3x2-x-11
↦ [2]:
↦    x6+2x5-9x4+x3-9x2+7x+7
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦    [3]:
↦       3
↦    [4]:
↦       4
↦    [5]:
↦       5
↦    [6]:
↦       6
ring R = (499,a),x,dp;
list d2 = 2,3a,5;
list e2 = (a-2), (9a2-8a), (a+10);
polyInterpolation(d2,e2);
↦ [1]:
↦    x2-3*x+(a)
↦ [2]:
↦    x3+(-3a-7)*x2+(21a+10)*x+(-30a)
↦ [3]:
↦    [1]:
↦       2
↦    [2]:
↦ (3a)
↦    [3]:
↦       5
```

D.4.9.3 modrationalInterpolation

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: modrationalInterpolation(D, E, vr[, D1, E1]); D list, E list, vr int, D1 list, E1 list

Return: a list L where r/t (r:=L[1], t:=L[2]) is a univariate rational function such that r(D[i])/t(D[i]) = E[i] (or equivalently r/t = g mod f, gcd(r,t)=gcd(f,t)=1 and deg(r) + deg(t) < deg(f)

Note: Optional parameters D1 and E1 can be provided to update the existing input, that is, to D1+D and E1+E. The rational function r/t is returned as an element in k(var(vr)), where k is a field. This procedure works only in characteristic zero. The elements in the first list must be distinct.

Example:

```
LIB "ffmodstd.lib";
ring rr=0,x,dp;
list D = 2,3,4,5,6,7,8,9,10;
list E = 8/35, 7/123, 22/1027, 4/391, 44/7779, 29/8405, 74/32771, 23/14763, 112/10000
modrationalInterpolation(D, E, 1);
↦ [1]:
↦    x2+x+2
↦ [2]:
↦    x5+3
ring R = 0, x, dp;
list D1 = 2,3,4,5,6;
list E1 = 8/35, 7/123, 22/1027, 4/391, 44/7779;
modrationalInterpolation(D1, E1, 1);
↦ [1]:
↦    35185737407/8965053161610x4-659760301271/8965053161610x3+8448757043/16\
   419511285x2-7177841332787/4482526580805x+1702354556926/896505316161
↦ [2]:
↦    1
list D = 7,8,9,10;
↦ // ** redefining D (list D = 7,8,9,10;) ./examples/modrationalInterpolati\
   on.sing:10
list E = 29/8405, 74/32771, 23/14763, 112/100003;
modrationalInterpolation(D, E, 1, D1, E1);
↦ [1]:
↦    x2+x+2
↦ [2]:
↦    x5+3
```

See also: Section D.4.9.1 [fareypoly], page 1081; Section D.4.9.2 [polyInterpolation], page 1082.

D.4.9.4 BerlekampMassey

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: BerlekampMassey(L, i[, M]); L list, i int, M list

Return: a list Tr where f:=Tr[1] is the minimal polynomial (w.r.t. the i-th variable) generated by the sequence (L[j]), 1<=j<= Tr[2], if the length of the sequence is long enough. In this case, the coefficients c_i of the polynomial f satisfy the relation -L[j+t] = c_0*L[j] + ... + c_{t-1}*L[j+t-1] for all j >=1 where t=deg(f).

Note: The procedure applies the Berlekamp/Massey algorithm to the sequence L[j] (elements from the field Q) for j>0 and returns a polynomial f. If the polynomial f splits into linear factors with no multiplicity greater than one, then we say that the length of the sequence L is long enough. If this polynomial does not split into linear factors, an optional parameter M = BerlekampMassey(L',i) can be provided to add more elements to the sequence.

References:

[1] E. Kaltofen and W.-s. Lee: Early termination in sparse interpolation algorithms. J. Symb. Comput. 36, 365-400 (2003).
[2] E. Kaltofen, W.-s. Lee and A. A. Lobo: Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. Proc. ISSAC (ISSAC '00), 192-201 (2000).

Example:

```
LIB "ffmodstd.lib";
ring rr=0,x,dp;
list L = 150,3204,79272,2245968;
list Tr = BerlekampMassey(L,1);
Tr[1];
↦ 117288/209x2-10662/209x+1
factorize(Tr[1]); //not linearly factored
↦ [1]:
↦    _[1]=1/209
↦    _[2]=117288x2-10662x+209
↦ [2]:
↦    1,1
list L1 = 70411680, 2352815424, 81496927872;
Tr = BerlekampMassey(L1,1,Tr); // increase the length of L by size(L1)
Tr[1];
↦ x3-66x2+1296x-7776
factorize(Tr[1]); //linearly factored and has distinct roots
↦ [1]:
↦    _[1]=1
↦    _[2]=x-36
↦    _[3]=x-18
↦    _[4]=x-12
↦ [2]:
↦    1,1,1,1
Tr[2]; //the length of the sequence required to generate Tr[1]
↦ 6
```

D.4.9.5 modberlekampMassey

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: modberlekampMassey(L [, i]); L list, i int

Return: The minimal polynomial f (w.r.t. the i-th variable) generated by the sequence (L[j]), j = 1, 2, ...

Note: The procedure first construct polynomials f and g of degrees size(L) and size(L)-1, respectively from the sequence L[j] (elements from the field Q) for j>0 as described in [1]. It then returns the denominator polynomial d obtained by applying the SINGULAR command Section D.4.9.1 [fareypoly], page 1081 to the input (g,f). If the ground ring has n variables, the procedure returns d in a polynomial ring k[var(i)] (k is a field) for some i<=n. In this case, an optional parameter i (default 0) can be provided.

References:

[1] Nadia Ben Atti, Gema M. Diaz-Toca and Henri Lombardi: The Berlekamp-Massey Algorithm Revisited, 2000.

Example:

```
LIB "ffmodstd.lib";
ring rr=0, (x,y,z), dp;
list L = 150,3204,79272,2245968, 70411680, 2352815424, 81496927872;
modberlekampMassey(L);// default w.r.t x
↦ x3-66x2+1296x-7776
```

```
modberlekampMassey(L,3);// returns an output in the ring Q[z]
↦ z3-66z2+1296z-7776
```

See also: Section D.4.9.4 [BerlekampMassey], page 1084.

D.4.9.6 sparseInterpolation

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: sparseInterpolation(Br, La, lpr, n[, m]); Br poly, La list, lpr list, n int, m int

Return: a polynomial B in the polynomial ring Q[var(n+1),...,var(n+size(lpr))] satisfying the relation La[i] = B(lpr[1]^i,...,lpr[size(lpr)]^i).

Note: The polynomial Br in Q[var(n)] is the minimal polynomial obtained by applying the SINGULAR command Section D.4.9.4 [BerlekampMassey], page 1084 to the sequence (La[j]), 1<=j<=size(La). By default the exponent i starts from 1. However, if the optional parameter m>=0 is provided, then it starts from m.
The list lpr must be a list of distinct primes.

Example:

```
LIB "ffmodstd.lib";
ring rr=0,(x,y),dp;
list lpr = 2,3; // assign 2 for x and 3 for y
list La = 150,3204,79272,2245968,70411680, 2352815424, 81496927872;
// La[i] = number(subst(f,y,lpr[1]^i,z,lpr[2]^i)); for f = x2y2+2x2y+5xy2 and i=1,..
poly Br = BerlekampMassey(La,1)[1];
Br;
↦ x3-66x2+1296x-7776
sparseInterpolation(Br,La,lpr,0); // reconstruct f default
↦ x2y2+2x2y+5xy2
La = 97,275,793,2315,6817;
// La[i] = number(subst(g,y,lpr[1]^i,z,lpr[2]^i)); for g = x+y and i=4,...,8
Br = BerlekampMassey(La,1)[1];
Br;
↦ x2-5x+6
sparseInterpolation(Br,La,lpr,0,4);
↦ x+y
```

See also: Section D.4.9.4 [BerlekampMassey], page 1084.

D.4.9.7 ffmodStd

Procedure from library `ffmodstd.lib` (see Section D.4.9 [ffmodstd_lib], page 1080).

Usage: ffmodStd(I [, d]); I ideal, d int

Return: Groebner basis of I over an algebraic function field

Note: An optional parameter d>0, a positive integer, can be provided for the procedure. It refers to a number of evaluation points to used.

Example:

```
LIB "ffmodstd.lib";
ring Ra=(0,a),(x,y,z),dp;
ideal I = (a^2+2)*x^2*y+a*y*z^2, x*z^2+(a+1)*x^2-a*y^2;
ffmodStd(I);
↦ _[1]=xz2+(a+1)*x2+(-a)*y2
↦ _[2]=(a2+2)*x2y+(a)*yz2
↦ _[3]=(a2+2)*yz4+(a4+4a2+4)*xy3+(a4+a3+2a2+2a)*y3+(a3+2a2+a)*yz2
ideal J = x^2*y+y*z^2, x*z^2+x^2-y^2;
ffmodStd(J);
↦ _[1]=xz2+x2-y2
↦ _[2]=x2y+yz2
↦ _[3]=yz4+xy3+y3+yz2
ring R1=(0,a,b),(x,y,z),dp;
ideal I = x^2*y^3*z+2*a*x*y*z^2+7*y^3,
x^2*y^4*z+(a-7b)*x^2*y*z^2-x*y^2*z^2+2*x^2*y*z-12*x+by,
(a2+b-2)*y^5*z+(a+5b)*x^2*y^2*z-b*x*y^3*z-x*y^3+y^4+2*a2*y^2*z,
a*x^2*y^2*z-x*y^3*z+3a*x*y*z^3+(-a+4)*y^3*z^2+4*z^2-bx;
ffmodStd(I);
↦ _[1]=12*x+(-b)*y
↦ _[2]=48*z2+(-b2)*y
↦ _[3]=y2z
↦ _[4]=y3
```

See also: Section D.4.20.2 [nfmodStd], page 1176.

D.4.10 grwalk_lib

Library: grwalk.lib

Purpose: Groebner Walk Conversion Algorithms

Author: I Made Sulandra

Procedures: See also: Section D.15.16 [rwalk_lib], page 2536; Section D.15.20 [swalk_lib], page 2550.

D.4.10.1 fwalk

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: fwalk(ideal i);
fwalk(ideal i, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal w.r.t. the
lexicographical ordering or a weighted-lex ordering, calculated via the fractal walk algorithm.

Example:

```
LIB "grwalk.lib";
ring r = 32003,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
fwalk(I);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.4.10.2 twalk

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: twalk(ideal i);
twalk(ideal i, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal w.r.t.
the ordering "(a(w),lp)" or "(a(1,0,...,0),lp)",
calculated via the Tran algorithm.

Example:

```
LIB "grwalk.lib";
ring r = 32003,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
twalk(I);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272.

D.4.10.3 awalk1

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: awalk1(ideal i);
awalk1(ideal i, int n);
awalk1(ideal i, int n, intvec v, intvec w);
awalk1(ideal i, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the first alternative algorithm from an ordering
"(a(v),lp)", "dp" or "Dp" to the ordering
"(a(w),lp)" or "(a(1,0,...,0),lp)"
with a perturbation degree n for the weight vector w.

Example:

```
LIB "grwalk.lib";
ring r = 32003,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
awalk1(I,3);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.4.10.4 awalk2

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: awalk2(ideal i);
awalk2(ideal i, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the second alternative algorithm from the ordering
"(a(v),lp)", "dp" or "Dp"
to the ordering "(a(w),lp)" or "(a(1,0,...,0),lp)".

Example:

```
   LIB "grwalk.lib";
   ring r = 32003,(z,y,x), lp;
   ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
   awalk2(I);
   ↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
      2x3-27yx3-27yx2-27x
   ↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
      9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
      x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
      x4+x5+2x4+x3
   ↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.4.10.5 pwalk

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: pwalk(int d, ideal i, int n1, int n2);
pwalk(int d, ideal i, int n1, int n2, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the perturbation walk algorithm from the ordering
"(a(v),lp)", "dp" or "Dp"
to the ordering "(a(w),lp)" or "(a(1,0,...,0),lp)"
with a perturbation degree n, m for v and w, resp.

Example:

```
   LIB "grwalk.lib";
   ring r = 32003,(z,y,x), lp;
```

```
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
pwalk(I,2,2);
↦ _[1]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[2]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.4.10.6 gwalk

Procedure from library `grwalk.lib` (see Section D.4.10 [grwalk_lib], page 1087).

Syntax: gwalk(ideal i);
gwalk(ideal i, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the improved Groebner walk algorithm from the ordering "(a(v),lp)", "dp" or "Dp"
to the ordering "(a(w),lp)" or "(a(1,0,...,0),lp)".

Example:

```
LIB "grwalk.lib";
//** compute a Groebner basis of I w.r.t. lp.
ring r = 32003,(z,y,x), lp;
ideal I = zy2+yx2+yx+3,
z3x+y3+zyx-yx2-yx-3,
z2yx3-y5+z2yx2+y3x2+y2x3+y3x+y2x2+3z2x+3y2+3yx,
zyx5+y6-y4x2-y3x3+2zyx4-y4x-y3x2+zyx3-3z2yx+3zx3-3y3-3y2x+3zx2,
yx7-y7+y5x2+y4x3+3yx6+y5x+y4x2+3yx5-6zyx3+yx4+3x5+3y4+3y3x-6zyx2+6x4+3x3-9zx;
gwalk(I);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.4.11 homolog_lib

Library: homolog.lib

Purpose: Procedures for Homological Algebra

Authors: Gert-Martin Greuel, greuel@mathematik.uni-kl.de,
Bernd Martin, martin@math.tu-cottbus.de
Christoph Lossen, lossen@mathematik.uni-kl.de

Procedures:

D.4.11.1 canonMap

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: canonMap(id); id= ideal/module,

Return: a list L, the kernel in two different representations and
the cokernel of the canonical map
M —> Ext^c_R(Ext^c_R(M,R),R) given by presentations
Here M is the R-module (R=basering) given by the presentation
defined by id, i.e. M=R/id resp. M=R^n/id
c is the codimension of M
L[1] is the preimage of the kernel in R resp. R^n
L[2] is a presentation of the kernel
L[3] is a presentation of the cokernel

Example:

```
LIB "homolog.lib";
ring s=0,(x,y),dp;
ideal i = x,y;
canonMap(i);
↦ [1]:
↦    _[1]=y*gen(1)
↦    _[2]=x*gen(1)
↦ [2]:
↦    _[1]=0
↦ [3]:
↦    _[1]=0
ring R = 0,(x,y,z,w),dp;
ideal I1 = x,y;
ideal I2 = z,w;
ideal I = intersect(I1,I2);
canonMap(I);
↦ [1]:
↦    _[1]=yw*gen(1)
↦    _[2]=xw*gen(1)
↦    _[3]=yz*gen(1)
↦    _[4]=xz*gen(1)
↦ [2]:
↦    _[1]=0
↦ [3]:
↦    _[1]=-w*gen(1)
↦    _[2]=-z*gen(1)
↦    _[3]=-y*gen(1)
↦    _[4]=-x*gen(1)
module M = syz(I);
canonMap(M);
↦ [1]:
```

```
↦    _[1]=z*gen(1)-w*gen(3)
↦    _[2]=z*gen(2)-w*gen(4)
↦    _[3]=x*gen(1)-y*gen(2)
↦    _[4]=x*gen(3)-y*gen(4)
↦ [2]:
↦    _[1]=0
↦ [3]:
↦    _[1]=yw*gen(1)
↦    _[2]=xw*gen(1)
↦    _[3]=yz*gen(1)
↦    _[4]=xz*gen(1)
ring S = 0,(x,y,z,t),Wp(3,4,5,1);
ideal I = x-t3,y-t4,z-t5;
ideal J = eliminate(I,t);
ring T = 0,(x,y,z),Wp(3,4,5);
ideal p = imap(S,J);
ideal p2 = p^2;
canonMap(p2);
↦ [1]:
↦    _[1]=x5*gen(1)-3x2yz*gen(1)+xy3*gen(1)+z3*gen(1)
↦    _[2]=x2z2*gen(1)-2xy2z*gen(1)+y4*gen(1)
↦    _[3]=x4z*gen(1)-x3y2*gen(1)-xyz2*gen(1)+y3z*gen(1)
↦    _[4]=x3yz*gen(1)-x2y3*gen(1)-xz3*gen(1)+y2z2*gen(1)
↦    _[5]=x4y2*gen(1)-4xy3z*gen(1)+2y5*gen(1)+z4*gen(1)
↦ [2]:
↦    _[1]=x*gen(1)
↦    _[2]=y*gen(1)
↦    _[3]=z*gen(1)
↦ [3]:
↦    _[1]=0
```

D.4.11.2 cup

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: cup(M,[,any,any]); M=module

Compute: cup-product Ext^1(M',M') x Ext^1(M',M') ---> Ext^2(M',M'), where M':=R^m/M, if M in R^m, R basering (i.e. M':=coker(matrix(M))).
If called with >= 2 arguments: compute symmetrized cup-product

Assume: all Ext's are finite dimensional

Return: - if called with 1 argument: matrix, the columns of the output present the coordinates of b_i&b_j with respect to a kbase of Ext^2, where b_1,b_2,... is a kbase of Ext^1 and & denotes cup product;
- if called with 2 arguments: matrix, the columns of the output present the coordinates of (1/2)(b_i&b_j + b_j&b_i) with respect to a kbase of Ext^2;
- if called with 3 arguments: list,
L[1] = matrix see above (symmetric case, for >=2 arguments)
L[2] = matrix of kbase of Ext^1
L[3] = matrix of kbase of Ext^2

Note: printlevel >=1; shows what is going on.
printlevel >=2; shows result in another representation.
For computing cupproduct of M itself, apply proc to syz(M)!

Example:

```
LIB "homolog.lib";
int p      = printlevel;
ring  rr   = 32003,(x,y,z),(dp,C);
ideal  I   = x4+y3+z2;
qring  o   = std(I);
module M   = [x,y,0,z],[y2,-x3,z,0],[z,0,-y,-x3],[0,z,x,-y2];
print(cup(M));
↦ 1,0, 0, 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,-1,0, 0,1,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,0, -1,0,0,0,0,0,1,0, 0,0,0,0,0,0,
↦ 0,0, 0, 1,0,0,1,0,0,-1,0,0,1,0,0,0
print(cup(M,1));
↦ 1,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,1,0,0,0,0,0,0
// 2nd EXAMPLE  (shows what is going on)
printlevel = 3;
ring   r   = 0,(x,y),(dp,C);
ideal  i   = x2-y3;
qring  q   = std(i);
module M   = [-x,y],[-y2,x];
print(cup(M));
↦ // vdim (Ext^1) = 2
↦ // kbase of Ext^1(M,M)
↦ //  - the columns present the kbase elements in Hom(F(1),F(0))
↦ //  - F(*) a free resolution of M
↦ -1,0,
↦ 0, y,
↦ 0, 1,
↦ -1,0
↦ // lift kbase of Ext^1:
↦ //  - the columns present liftings of kbase elements into Hom(F(2),F(1))
↦ //  - F(*) a free resolution of M
↦ 1,0,
↦ 0,y,
↦ 0,1,
↦ 1,0
↦ // vdim (Ext^2) = 2
↦ // kbase of Ext^2(M,M)
↦ //  - the columns present the kbase elements in Hom(F(2),F(0))
↦ //  - F(*) is a free resolution of M
↦ -1,0,
↦ 0, y,
↦ 0, 1,
↦ -1,0
↦ // matrix of cup-products (in Ext^2)
↦ -1,0, 0,y,
↦ 0, -y,y,0,
↦ 0, -1,1,0,
↦ -1,0, 0,y
↦ ////// end level 2 //////
↦ // the associated matrices of the bilinear mapping 'cup'
↦ // corresponding to the kbase elements of Ext^2(M,M) are shown,
↦ //  i.e. the rows of the final matrix are written as matrix of
↦ //  a bilinear form on Ext^1 x Ext^1
↦ //-----component 1:
↦ 1,0,
↦ 0,0
↦ //-----component 2:
↦ 0,-1,
↦ 1,0
↦ ////// end level 3 //////
↦ 1,0, 0,0,
↦ 0,-1,1,0
printlevel = p;
```

D.4.11.3 cupproduct

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: cupproduct(M,N,P,p,q[,any]); M,N,P modules, p,q integers

Compute: cup-product Ext^p(M',N') x Ext^q(N',P') —> Ext^(p+q)(M',P'), where M':=R^m/M, if M in R^m, R basering (i.e. M':=coker(matrix(M)))

Assume: all Ext's are of finite dimension

Return: - if called with 5 arguments: matrix of the associated linear map Ext^p (tensor) Ext^q –> Ext^(p+q), i.e. the columns of <matrix> present the coordinates of the cup products (b_i & c_j) with respect to a kbase of Ext^p+q (b_i resp. c_j are the chosen bases of Ext^p, resp. Ext^q).
- if called with 6 arguments: list L,
L[1] = matrix (see above)
L[2] = matrix of kbase of Ext^p(M',N')
L[3] = matrix of kbase of Ext^q(N',P')
L[4] = matrix of kbase of Ext^p+q(N',P')

Note: printlevel >=1; shows what is going on.
printlevel >=2; shows the result in another representation.
For computing the cupproduct of M,N itself, apply proc to syz(M), syz(N)!

Example:

```
LIB "homolog.lib";
int p      = printlevel;
ring  rr   = 32003,(x,y,z),(dp,C);
ideal  I   = x4+y3+z2;
qring  o   = std(I);
module M   = [x,y,0,z],[y2,-x3,z,0],[z,0,-y,-x3],[0,z,x,-y2];
print(cupproduct(M,M,M,1,3));
↦ 1,0, 0, 0,0,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,-1,0, 0,1,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,0, -1,0,0,0,0,0,1,0, 0,0,0,0,0,0,
↦ 0,0, 0, 1,0,0,1,0,0,-1,0,0,1,0,0,0
printlevel = 3;
list l     = (cupproduct(M,M,M,1,3,"any"));
↦ // vdim Ext(M,N) = 4
↦ // kbase of Ext^p(M,N)
↦ //  - the columns present the kbase elements in Hom(F(p),G(0))
↦ //  - F(*),G(*) are free resolutions of M and N
↦ 0, 0, 1, 0,
↦ 0, y, 0, 0,
↦ 1, 0, 0, 0,
↦ 0, 0, 0, y,
↦ 0, -1,0, 0,
↦ 0, 0, x2,0,
↦ 0, 0, 0, -x2,
↦ 1, 0, 0, 0,
↦ 0, 0, 0, -1,
↦ -1,0, 0, 0,
↦ 0, 1, 0, 0,
↦ 0, 0, 1, 0,
↦ -1,0, 0, 0,
↦ 0, 0, 0, x2y,
↦ 0, 0, x2,0,
↦ 0, -y,0, 0
↦ // vdim Ext(N,P) = 4
↦ // kbase of Ext(N,P):
↦ 0, 0, 1,  0,
↦ 0, 0, 0,  y,
↦ 1, 0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, -1,0,  0,
↦ 1, 0, 0,  0,
↦ 0, 0, 0,  -x2,
↦ 0, 0, -x2,0,
↦ 0, 0, 0,  -1,
↦ 0, 0, 1,  0,
↦ 0, 1, 0,  0,
↦ 1, 0, 0,  0,
↦ -1,0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, 0, x2, 0,
↦ 0, 0, 0,  -x2y
↦ // kbase of Ext^q(N,P)
↦ //  - the columns present the kbase elements in Hom(G(q),H(0))
↦ //  - G(*),H(*) are free resolutions of N and P
↦ 0, 0, 1,  0,
↦ 0, 0, 0,  y,
↦ 1, 0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, -1,0,  0,
↦ 1, 0, 0,  0,
↦ 0, 0, 0,  -x2,
↦ 0, 0, -x2,0,
↦ 0, 0, 0,  -1,
↦ 0, 0, 1,  0,
↦ 0, 1, 0,  0,
↦ 1, 0, 0,  0,
↦ -1,0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, 0, x2, 0,
↦ 0, 0, 0,  -x2y
↦ // vdim Ext(M,P) = 4
↦ // kbase of Ext^p+q(M,P)
↦ //  - the columns present the kbase elements in Hom(F(p+q),H(0))
↦ //  - F(*),H(*) are free resolutions of M and P
↦ 0, 0, 1,  0,
↦ 0, 0, 0,  y,
↦ 1, 0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, -1,0,  0,
↦ 1, 0, 0,  0,
↦ 0, 0, 0,  -x2,
↦ 0, 0, -x2,0,
↦ 0, 0, 0,  -1,
↦ 0, 0, 1,  0,
↦ 0, 1, 0,  0,
↦ 1, 0, 0,  0,
↦ -1,0, 0,  0,
↦ 0, -y,0,  0,
↦ 0, 0, x2, 0,
↦ 0, 0, 0,  -x2y
↦ // lifting of kbase of Ext^p(M,N)
↦ //  - the columns present liftings of kbase elements in Hom(F(p+q),G(q))
↦ 1,0, 0, 0,
↦ 0,-y,0, 0,
↦ 0,0, x2,0,
↦ 0,0, 0, x2y,
↦ 0,1, 0, 0,
↦ 1,0, 0, 0,
↦ 0,0, 0, -x2,
↦ 0,0, x2,0,
↦ 0,0, -1,0,
↦ 0,0, 0, y,
↦ 1,0, 0, 0,
↦ 0,y, 0, 0,
↦ 0,0, 0, -1,
↦ 0,0, -1,0,
↦ 0,-1,0, 0,
↦ 1,0, 0, 0
↦ // matrix of cup-products (in Ext^p+q)
↦ 0, 0, -1, 0,   0, 0, 0,   y,   1,  0,  0,  0,  0,   y,   0,  0,
↦ 0, 0, 0,  y,   0, 0, y,   0,   0,  -y, 0,  0,  y,   0,   0,  0,
↦ 1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ 0, y, 0,  0,   -y,0, 0,   0,   0,  0,  0,  x2y,0,   0,   x2y,0,
↦ 0, 1, 0,  0,   -1,0, 0,   0,   0,  0,  0,  x2, 0,   0,   x2, 0,
↦ 1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ 0, 0, 0,  -x2, 0, 0, -x2, 0,   0,  x2, 0,  0,  -x2, 0,   0,  0,
↦ 0, 0, x2, 0,   0, 0, 0,   -x2y,-x2,0,  0,  0,  0,   -x2y,0,  0,
↦ 0, 0, 0,  -1,  0, 0, -1,  0,   0,  1,  0,  0,  -1,  0,   0,  0,
↦ 0, 0, -1, 0,   0, 0, 0,   y,   1,  0,  0,  0,  0,   y,   0,  0,
↦ 0, -1,0,  0,   1, 0, 0,   0,   0,  0,  0,  -x2,0,   0,   -x2,0,
↦ 1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ -1,0, 0,  0,   0, -y,0,   0,   0,  0,  -x2,0,  0,   0,   0,  x2y,
↦ 0, y, 0,  0,   -y,0, 0,   0,   0,  0,  0,  x2y,0,   0,   x2y,0,
↦ 0, 0, -x2,0,   0, 0, 0,   x2y, x2, 0,  0,  0,  0,   x2y, 0,  0,
↦ 0, 0, 0,  -x2y,0, 0, -x2y,0,   0,  x2y,0,  0,  -x2y,0,   0,  0
↦ ////// end level 2 //////
↦ // the associated matrices of the bilinear mapping 'cup'
↦ // corresponding to the kbase elements of Ext^p+q(M,P) are shown,
↦ //  i.e. the rows of the final matrix are written as matrix of
↦ //  a bilinear form on Ext^p x Ext^q
↦ //----component 1:
↦ 1,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
↦ //----component 2:
↦ 0,-1,0,0,
↦ 1,0, 0,0,
↦ 0,0, 0,0,
↦ 0,0, 0,0
↦ //----component 3:
↦ 0,0,-1,0,
↦ 0,0,0, 0,
↦ 1,0,0, 0,
↦ 0,0,0, 0
↦ //----component 4:
↦ 0,0, 0,1,
↦ 0,0, 1,0,
↦ 0,-1,0,0,
↦ 1,0, 0,0
↦ ////// end level 3 //////
show(l[1]);show(l[2]);
↦ // matrix, 4x16
↦ 1,0, 0, 0,0,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,-1,0, 0,1,0,0,0,0,0, 0,0,0,0,0,0,
↦ 0,0, -1,0,0,0,0,0,1,0, 0,0,0,0,0,0,
↦ 0,0, 0, 1,0,0,1,0,0,-1,0,0,1,0,0,0
↦ // matrix, 16x4
↦ 0, 0, 1, 0,
↦ 0, y, 0, 0,
↦ 1, 0, 0, 0,
↦ 0, 0, 0, y,
↦ 0, -1,0, 0,
↦ 0, 0, x2,0,
↦ 0, 0, 0, -x2,
↦ 1, 0, 0, 0,
↦ 0, 0, 0, -1,
↦ -1,0, 0, 0,
↦ 0, 1, 0, 0,
↦ 0, 0, 1, 0,
↦ -1,0, 0, 0,
↦ 0, 0, 0, x2y,
↦ 0, 0, x2,0,
↦ 0, -y,0, 0
printlevel = p;
```

D.4.11.4 depth

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: depth(M,[I]); M module, I ideal

Return: int,
- if called with 1 argument: the depth of M'=coker(M) w.r.t. the ideal generated by the variables in the basering
(the maximal ideal, if the ring is local)
- if called with 2 arguments: the depth of M'=coker(M) w.r.t. the ideal I.

Note: Not checked: if I*M'==M', depth is infinity.

Note: procedure makes use of KoszulHomology.

Example:

```
LIB "homolog.lib";
ring R=0,(x,y,z),dp;
ideal I=x2,xy,yz;
module M=0;
depth(M,I);  // depth(<x2,xy,yz>,Q[x,y,z])
↦ 2
M=[1];
depth(M);  // depth(0)
↦ 3
ring r=0,(x,y,z),ds;  // local ring
matrix M[2][2]=x,xy,1+yz,0;
print(M);
↦ x,   xy,
↦ 1+yz,0
depth(M);     // depth(maxideal,coker(M))
↦ 2
ideal I=x;
depth(M,I);   // depth(<x>,coker(M))
↦ 0
I=x+z;
depth(M,I);   // depth(<x+z>,coker(M))
↦ 1
```

D.4.11.5 Ext_R

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: Ext_R(v,M[,p]); v int resp. intvec , M module, p int

Compute: A presentation of Ext^k(M',R); for k=v[1],v[2],..., M'=coker(M). Let

```
0 <-- M' <-- F0 <-M-- F1 <-- F2 <-- ...
```

be a free resolution of M'. If

```
0 --> F0* -A1-> F1* -A2-> F2* -A3-> ...
```

is the dual sequence, Fi*=Hom(Fi,R), then Ext^k = ker(Ak+1)/im(Ak) is presented as in the following exact sequences:

```
R^p --syz(Ak+1)-> Fk* ---Ak+1---->  Fk+1* ,
R^q ----Ext^k---> R^p --syz(Ak+1)-> Fk*/im(Ak).
```

Hence, Ext^k=modulo(syz(Ak+1),Ak) presents Ext^k(M',R).

Return: - module Ext, a presentation of Ext^k(M',R) if v is of type int
- a list of Ext^k (k=v[1],v[2],...) if v is of type intvec.
- In case of a third argument of type int return a list l:
  - l[1] = module Ext^k resp. list of Ext^k
  - l[2] = SB of Ext^k resp. list of SB of Ext^k
  - l[3] = matrix resp. list of matrices, each representing a kbase of Ext^k (if finite dimensional)

Display: printlevel >=0: (affine) dimension of Ext^k for each k (default) printlevel >=1: Ak, Ak+1 and kbase of Ext^k in Fk*

Note: In order to compute Ext^k(M,R) use the command Ext_R(k,syz(M));
By default, the procedure uses the `mres` command. If called with the additional parameter `"sres"`, the `sres` command is used instead.
If the attribute `"isHomog"` has been set for the input module, it is also set for the returned module (accordingly).

Example:

```
LIB "homolog.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 0,(x,y,z),dp;
ideal i    = x2y,y2z,z3x;
module E   = Ext_R(1,i);    //computes Ext^1(r/i,r)
↦ // Computing Ext^1:
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of M,
↦ // then F1*-->F2* is given by:
↦ x2, -yz,0,
↦ 0,  z3, -xy,
↦ xz2,0,  -y2
↦ // and F0*-->F1* is given by:
↦ y2z,
↦ x2y,
↦ xz3
↦
↦ // dimension of Ext^1:  -1
↦
is_zero(E);
↦ 1
qring R    = std(x2+yz);
intvec v   = 0,2;
printlevel = 2;             //shows what is going on
ideal i    = x,y,z;         //computes Ext^i(r/(x,y,z),r/(x2+yz)), i=0,2
list L     = Ext_R(v,i,1);  //over the qring R=r/(x2+yz), std and kbase
↦ // Computing Ext^0:
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of M,
↦ // then F0*-->F1* is given by:
↦ z,
↦ y,
↦ x
↦ // and F-1*-->F0* is given by:
↦ 0
↦
↦ // dimension of Ext^0:  -1
↦
↦ // columns of matrix are kbase of Ext^0 in F0*:
↦ 0
↦
↦ // Computing Ext^2:
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of M,
↦ // then F2*-->F3* is given by:
↦ x,-y,z, 0,
↦ z,x, 0, z,
↦ 0,0, x, y,
↦ 0,0, -z,x
↦ // and F1*-->F2* is given by:
↦ y,-z,0,
↦ x,0, -z,
↦ 0,x, -y,
↦ 0,z, x
↦
↦ // dimension of Ext^2:  0
↦ // vdim of Ext^2:       1
↦
↦ // columns of matrix are kbase of Ext^2 in F2*:
↦ x,
↦ -z,
↦ 0,
↦ 0
↦
printlevel = p;
```

D.4.11.6 Ext

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: Ext(v,M,N[,any]); v int resp. intvec, M,N modules

Compute: A presentation of Ext^k(M',N'); for k=v[1],v[2],... where M'=coker(M) and N'=coker(N). Let

```
0 <-- M' <-- F0 <-M-- F1 <-- F2 <--... ,
0 <-- N' <-- G0 <--N- G1
```

be a free resolution of M', resp. a presentation of N'. Consider the commutative diagram

```
             0                  0                  0
             |^                 |^                 |^
--> Hom(Fk-1,N') -Ak-> Hom(Fk,N') -Ak+1-> Hom(Fk+1,N')
             |^                 |^                 |^
--> Hom(Fk-1,G0) -Ak-> Hom(Fk,G0) -Ak+1-> Hom(Fk+1,G0)
                                |^                 |^
                                |C                 |B
                       Hom(Fk,G1) ------> Hom(Fk+1,G1)

      (Ak,Ak+1 induced by M and B,C induced by N).
```

Let K=modulo(Ak+1,B), J=module(Ak)+module(C) and Ext=modulo(K,J), then we have exact sequences

```
         R^p --K-> Hom(Fk,G0) --Ak+1-> Hom(Fk+1,G0)/im(B),

         R^q -Ext-> R^p --K-> Hom(Fk,G0)/(im(Ak)+im(C)).
```

Hence, Ext presents Ext^k(M',N').

Return: - module Ext, a presentation of Ext^k(M',N') if v is of type int
- a list of Ext^k (k=v[1],v[2],...) if v is of type intvec.
- In case of a third argument of any type return a list l:

l[1] = module Ext/list of Ext^k
l[2] = SB of Ext/list of SB of Ext^k
l[3] = matrix/list of matrices, each representing a kbase of Ext^k
(if finite dimensional)

Display: printlevel >=0: dimension, vdim of Ext^k for each k (default).
printlevel >=1: matrices Ak, Ak+1 and kbase of Ext^k in Hom(Fk,G0) (if finite dimensional)

Note: In order to compute Ext^k(M,N) use the command Ext(k,syz(M),syz(N)); or: list P=mres(M,2); list Q=mres(N,2); Ext(k,P[2],Q[2]);

Example:

```
LIB "homolog.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 0,(x,y),dp;
ideal i    = x2-y3;
ideal j    = x2-y5;
list E     = Ext(0..2,i,j);    // Ext^k(r/i,r/j) for k=0,1,2 over r
↦ // Computing Ext^0 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F0,G0)-->Hom(F1,G0) is given by:
↦ y3-x2
↦ // and Hom(F-1,G0) + Hom(F0,G1)-->Hom(F0,G0) is given by:
↦ 0,-y5+x2
↦
↦ // dimension of Ext^0:  -1
↦
↦ // Computing Ext^1 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F1,G0)-->Hom(F2,G0) is given by:
↦ 0
↦ // and Hom(F0,G0) + Hom(F1,G1)-->Hom(F1,G0) is given by:
↦ y3-x2,-y5+x2
↦
↦ // dimension of Ext^1:  0
↦ // vdim of Ext^1:       10
↦
↦ // Computing Ext^2 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F2,G0)-->Hom(F3,G0) is given by:
↦ 1
↦ // and Hom(F1,G0) + Hom(F2,G1)-->Hom(F2,G0) is given by:
↦ 0,-y5+x2
↦ 
↦ // dimension of Ext^2:  -1
↦ 
qring R    = std(i);
ideal j    = fetch(r,j);
module M   = [-x,y],[-y2,x];
printlevel = 2;
module E1  = Ext(1,M,j);       // Ext^1(R^2/M,R/j) over R=r/i
↦ // Computing Ext^1 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F1,G0)-->Hom(F2,G0) is given by:
↦ x, -y,
↦ y2,-x
↦ // and Hom(F0,G0) + Hom(F1,G1)-->Hom(F1,G0) is given by:
↦ x, -y,-y5+x2,0,
↦ y2,-x,0,     -y5+x2
↦ 
↦ // dimension of Ext^1:  -1
↦ 
list l     = Ext(4,M,M,1);     // Ext^4(R^2/M,R^2/M) over R=r/i
↦ // Computing Ext^4 (help Ext; gives an explanation):
↦ // Let 0<--coker(M)<--F0<--F1<--F2<--... be a resolution of coker(M),
↦ // and 0<--coker(N)<--G0<--G1 a presentation of coker(N),
↦ // then Hom(F4,G0)-->Hom(F5,G0) is given by:
↦ x, -y,0, 0,
↦ y2,-x,0, 0,
↦ 0, 0, x, -y,
↦ 0, 0, y2,-x
↦ // and Hom(F3,G0) + Hom(F4,G1)-->Hom(F4,G0) is given by:
↦ x, -y,0, 0, -x,0, -y2,0,
↦ y2,-x,0, 0, 0, -x,0,  -y2,
↦ 0, 0, x, -y,y, 0, x,  0,
↦ 0, 0, y2,-x,0, y, 0,  x
↦ 
↦ // dimension of Ext^4:  0
↦ // vdim of Ext^4:       2
↦ 
↦ // columns of matrix are kbase of Ext^4 in Hom(F4,G0)
↦ 1,0,
↦ 0,y,
↦ 0,1,
↦ 1,0
↦ 
↦ // element 1 of kbase of Ext^4 in Hom(F4,G0)
↦ // as matrix: F4-->G0
↦ 1,0,
↦ 0,1
↦ // element 2 of kbase of Ext^4 in Hom(F4,G0)
↦ // as matrix: F4-->G0
↦ 0,y,
↦ 1,0
↦
printlevel = p;
```

D.4.11.7 fitting

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: fitting (M,n); M module, n int

Return: ideal, (standard basis of) n-th Fitting ideal of M'=coker(M).

Example:

```
LIB "homolog.lib";
ring R=0,x(0..4),dp;
matrix M[2][4]=x(0),x(1),x(2),x(3),x(1),x(2),x(3),x(4);
print(M);
↦ x(0),x(1),x(2),x(3),
↦ x(1),x(2),x(3),x(4)
fitting(M,-1);
↦ _[1]=0
fitting(M,0);
↦ _[1]=x(3)^2-x(2)*x(4)
↦ _[2]=x(2)*x(3)-x(1)*x(4)
↦ _[3]=x(1)*x(3)-x(0)*x(4)
↦ _[4]=x(2)^2-x(0)*x(4)
↦ _[5]=x(1)*x(2)-x(0)*x(3)
↦ _[6]=x(1)^2-x(0)*x(2)
fitting(M,1);
↦ _[1]=x(4)
↦ _[2]=x(3)
↦ _[3]=x(2)
↦ _[4]=x(1)
↦ _[5]=x(0)
fitting(M,2);
↦ _[1]=1
```

D.4.11.8 flatteningStrat

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: flatteningStrat(M); M module

Return: list of ideals.
The list entries L[1],...,L[r] describe the flattening stratification of M'=coker(M): setting L[0]=0, L[r+1]=1, the flattening stratification is given by the open sets Spec(A/V(L[i-1])) \ V(L[i]), i=1,...,r+1 (A = basering).

Note: for more information see the book 'A Singular Introduction to Commutative Algebra' (by Greuel/Pfister, Springer 2002).

Example:

```
LIB "homolog.lib";
ring A = 0,x(0..4),dp;
```

```
// presentation matrix:
matrix M[2][4] = x(0),x(1),x(2),x(3),x(1),x(2),x(3),x(4);
list L = flatteningStrat(M);
L;
↦ [1]:
↦    _[1]=x(3)^2-x(2)*x(4)
↦    _[2]=x(2)*x(3)-x(1)*x(4)
↦    _[3]=x(1)*x(3)-x(0)*x(4)
↦    _[4]=x(2)^2-x(0)*x(4)
↦    _[5]=x(1)*x(2)-x(0)*x(3)
↦    _[6]=x(1)^2-x(0)*x(2)
↦ [2]:
↦    _[1]=x(4)
↦    _[2]=x(3)
↦    _[3]=x(2)
↦    _[4]=x(1)
↦    _[5]=x(0)
```

D.4.11.9 Hom

Procedure from library homolog.lib (see Section D.4.11 [homolog_lib], page 1090).

Usage: Hom(M,N,[any]); M,N=modules

Compute: A presentation of Hom(M',N'), M'=coker(M), N'=coker(N) as follows: let

```
        F1 --M-> F0 -->M' --> 0,    G1 --N-> G0 --> N' --> 0
```

be presentations of M' and N'. Consider

```
                                       0                0
                                       |^               |^
            0 --> Hom(M',N') ----> Hom(F0,N') ----> Hom(F1,N')
                                       |^               |^
       (A:  induced by M)          Hom(F0,G0) --A-> Hom(F1,G0)
                                       |^               |^
       (B,C:induced by N)              |C               |B
                                   Hom(F0,G1) ----> Hom(F1,G1)
```

Let D=modulo(A,B) and Hom=modulo(D,C), then we have exact sequences

```
        R^p  --D-> Hom(F0,G0) --A-> Hom(F1,G0)/im(B),

      R^q -Hom-> R^p --D-> Hom(F0,G0)/im(C) --A-> Hom(F1,G0)/im(B).
```

Hence Hom presents Hom(M',N')

Return: module Hom, a presentation of Hom(M',N'), resp., in case of 3 arguments, a list l (of size <=3):

- l[1] = Hom
- l[2] = SB of Hom
- l[3] = kbase of coker(Hom) (if finite dimensional, not 0),
  represented by elements in Hom(F0,G0) via mapping D

Display: printlevel >=0: (affine) dimension of Hom (default)
printlevel >=1: D and C and kbase of coker(Hom) in Hom(F0,G0)
printlevel >=2: elements of kbase of coker(Hom) as matrix :F0–>G0

Note: DISPLAY is as described only for a direct call of 'Hom'. Calling 'Hom' from another proc has the same effect as decreasing printlevel by 1.

Example:

```
LIB "homolog.lib";
int p     = printlevel;
printlevel= 1;   //in 'example proc' printlevel has to be increased by 1
ring r    = 0,(x,y),dp;
ideal i   = x2-y3,xy;
qring q   = std(i);
ideal i   = fetch(r,i);
module M  = [-x,y],[-y2,x],[x3];
module H  = Hom(M,i);
↦ // dimension of Hom:  0
↦ // vdim of Hom:       5
↦
↦ // given  F1 --M-> F0 -->M'--> 0 and  G1 --N-> G0 -->N'--> 0,
↦ // show D = ker( Hom(F0,G0) --> Hom(F1,G0)/im(Hom(F1,G1)->Hom(F1,G0)) )
↦ y,x, 0,
↦ x,y2,x2
↦ // show C = im ( Hom(F0,G1) --> Hom(F0,G0) )
↦ -y3+x2,0,     xy,0,
↦ 0,     -y3+x2,0, xy
↦
print(H);
↦ 0, x, 0,y2,0,
↦ y, 0, 0,-x,x2,
↦ -1,-1,x,0, 0
printlevel= 2;
list L    = Hom(M,i,1);"";
↦ // dimension of Hom:  0
↦ // vdim of Hom:       5
↦
↦ // given  F1 --M-> F0 -->M'--> 0 and  G1 --N-> G0 -->N'--> 0,
↦ // show D = ker( Hom(F0,G0) --> Hom(F1,G0)/im(Hom(F1,G1)->Hom(F1,G0)) )
↦ y,x, 0,
↦ x,y2,x2
↦ // show C = im ( Hom(F0,G1) --> Hom(F0,G0) )
↦ -y3+x2,0,     xy,0,
↦ 0,     -y3+x2,0, xy
↦
↦ // element 1 of kbase of Hom in Hom(F0,G0) as matrix: F0-->G0:
↦ y2,xy
↦ // element 2 of kbase of Hom in Hom(F0,G0) as matrix: F0-->G0:
↦ y,x
↦ // element 3 of kbase of Hom in Hom(F0,G0) as matrix: F0-->G0:
↦ x2,xy2
↦ // element 4 of kbase of Hom in Hom(F0,G0) as matrix: F0-->G0:
↦ x,y2
↦ // element 5 of kbase of Hom in Hom(F0,G0) as matrix: F0-->G0:
↦ 0,x2
↦
printlevel=1;
```

```
ring s    = 3,(x,y,z),(c,dp);
ideal i   = jacob(ideal(x2+y5+z4));
qring rq=std(i);
matrix M[2][2]=xy,x3,5y,4z,x2;
matrix N[3][2]=x2,x,y3,3xz,x2z,z;
print(M);
↦ xy,x3,
↦ -y,z
print(N);
↦ x2, x,
↦ y3, 0,
↦ x2z,z
list l=Hom(M,N,1);
↦ // dimension of Hom:  0
↦ // vdim of Hom:       16
↦
↦ // given  F1 --M-> F0 -->M'--> 0 and  G1 --N-> G0 -->N'--> 0,
↦ // show D = ker( Hom(F0,G0) --> Hom(F1,G0)/im(Hom(F1,G1)->Hom(F1,G0)) )
↦ 0,0, 0,0, 0,   0,0,   1,
↦ 0,0, 0,0, 0,   0,y3z2,0,
↦ 0,0, 0,0, 0,   1,0,   0,
↦ 0,0, 0,y3,y2z2,0,0,   0,
↦ 0,0, 1,0, 0,   0,0,   0,
↦ z,y3,0,0, 0,   0,0,   0
↦ // show C = im ( Hom(F0,G1) --> Hom(F0,G0) )
↦ x2, 0,  x,0,
↦ 0,  x2, 0,x,
↦ y3, 0,  0,0,
↦ 0,  y3, 0,0,
↦ x2z,0,  z,0,
↦ 0,  x2z,0,z
↦
↦ // columns of matrix are kbase of Hom in Hom(F0,G0)
↦ 0, 0, 0, 0,0,0,   0,   0,  0, 0,  0, 0,0, 0,0,0,
↦ 0, 0, 0, 0,0,0,   0,   0,  0, 0,  0, 0,0, 0,0,y3z2,
↦ 0, 0, 0, 0,0,0,   y2z2,yz2,z2,y2z,yz,z,y2,y,1,0,
↦ 0, 0, 0, 0,0,y2z2,0,   0,  0, 0,  0, 0,0, 0,0,0,
↦ 0, y3,y2,y,1,0,   0,   0,  0, 0,  0, 0,0, 0,0,0,
↦ y3,0, 0, 0,0,0,   0,   0,  0, 0,  0, 0,0, 0,0,0
printlevel = p;
```

D.4.11.10 homology

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: homology(A,B,M,N);

Compute: Let M and N be submodules of R^m and R^n presenting M'=R^m/M, N'=R^n/N (R=basering) and let A,B matrices inducing maps

```
R^k --A--> R^m --B--> R^n.
```

Compute a presentation of the module

```
ker(B)/im(A) := ker(M'/im(A) --B--> N'/im(BM)+im(BA)).
```

If B induces a map M'–>N' (i.e BM=0) and if im(A) is contained in ker(B) (that is, BA=0) then ker(B)/im(A) is the homology of the complex

```
                    R^k--A-->M'--B-->N'.
```

Return: module H, a presentation of ker(B)/im(A).

Note: homology returns a free module of rank m if ker(B)=im(A).

Example:

```
LIB "homolog.lib";
ring r;
ideal id=maxideal(4); // GB
qring qr=id;
module N=maxideal(3)*freemodule(2);
module M=maxideal(2)*freemodule(2);
module B=[2x,0],[x,y],[z2,y];
module A=M;
module H=homology(A,B,M,N);
H=std(H);
// dimension of homology:
dim(H);
↦ 0
// vector space dimension:
vdim(H);
↦ 19
ring s=0,x,ds;
qring qs=std(x4);
module A=[x];
module B=A;
module M=[x3];
module N=M;
homology(A,B,M,N);
↦ _[1]=gen(1)
```

D.4.11.11 isCM

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: isCM(M); M module

Return: 1 if M'=coker(M) is Cohen-Macaulay;
0 if this is not the case.

Assume: basering is local.

Example:

```
LIB "homolog.lib";
ring R=0,(x,y,z),ds;  // local ring R = Q[x,y,z]_<x,y,z>
module M=xz,yz,z2;
isCM(M);             // test if R/<xz,yz,z2> is Cohen-Macaulay
↦ 0
M=x2+y2,z7;          // test if R/<x2+y2,z7> is Cohen-Macaulay
isCM(M);
↦ 1
```

D.4.11.12 isFlat

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: isFlat(M); M module

Return: 1 if M'=coker(M) is flat;
0 if this is not the case.

Example:

```
LIB "homolog.lib";
ring A = 0,(x,y),dp;
matrix M[3][3] = x-1,y,x,x,x+1,y,x2,xy+x+1,x2+y;
print(M);
↦ x-1,y,     x,
↦ x,  x+1,   y,
↦ x2, xy+x+1,x2+y
isFlat(M);             // coker(M) is not flat over A=Q[x,y]
↦ 0
qring B = std(x2+x-y);   // the ring B=Q[x,y]/<x2+x-y>
matrix M = fetch(A,M);
isFlat(M);             // coker(M) is flat over B
↦ 1
setring A;
qring C = std(x2+x+y);   // the ring C=Q[x,y]/<x2+x+y>
matrix M = fetch(A,M);
isFlat(M);             // coker(M) is not flat over C
↦ 0
```

D.4.11.13 isLocallyFree

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: isLocallyFree(M,r); M module, r int

Return: 1 if M'=coker(M) is locally free of constant rank r;
0 if this is not the case.

Example:

```
LIB "homolog.lib";
ring R=0,(x,y,z),dp;
matrix M[2][3];     // the presentation matrix
M=x-1,y-1,z,y-1,x-2,x;
ideal I=fitting(M,0); // 0-th Fitting ideal of coker(M)
qring Q=I;
matrix M=fetch(R,M);
isLocallyFree(M,1); // as R/I-module, coker(M) is locally free of rk 1
↦ 1
isLocallyFree(M,0);
↦ 0
```

D.4.11.14 isReg

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: isReg(I,M); I ideal, M module

Return: 1 if given (ordered) list of generators for I is coker(M)-sequence;
0 if this is not the case.

Example:

```
LIB "homolog.lib";
ring R = 0,(x,y,z),dp;
ideal I = x*(y-1),y,z*(y-1);
isReg(I,0);             // given list of generators is Q[x,y,z]-sequence
↦ 1
I = x*(y-1),z*(y-1),y;  // change sorting of generators
isReg(I,0);
↦ 0
ring r = 0,(x,y,z),ds;  // local ring
ideal I=fetch(R,I);
isReg(I,0);             // result independent of sorting of generators
↦ 1
```

D.4.11.15 hom_kernel

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: hom_kernel(A,M,N);

Compute: Let M and N be submodules of R^m and R^n, presenting M'=R^m/M, N'=R^n/N (R=basering), and let A:R^m–>R^n be a matrix inducing a map A':M'–>N'. Then ker(A,M,N); computes a presentation K of ker(A') as in the commutative diagram:

$$\begin{tikzcd}
\mathrm{ker}(A') \arrow[r] & M' \arrow[r, "A'"] & N' \\
R^r \arrow[u] \arrow[r] & R^m \arrow[u] \arrow[r, "A"] & R^n \arrow[u] \\
R^s \arrow[u, "K"] \arrow[r] & R^p \arrow[u, "M"] \arrow[r] & R^q \arrow[u, "N"]
\end{tikzcd}$$

Return: module K, a presentation of ker(A':coker(M)->coker(N)).

Example:

```
LIB "homolog.lib";
ring r;
module N=[2x,x],[0,y];
module M=maxideal(1)*freemodule(2);
matrix A[2][3]=2x,0,x,y,z2,y;
module K=hom_kernel(A,M,N);
// dimension of kernel:
dim(std(K));
↦ 3
// vector space dimension of kernel:
vdim(std(K));
↦ -1
print(K);
↦ 0,0,0,
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
```

D.4.11.16 kohom

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: kohom(A,k); A=matrix, k=integer

Return: matrix Hom(R^k,A), i.e. let A be a matrix defining a map F1–>F2 of free R-modules, then the matrix of Hom(R^k,F1)–>Hom(R^k,F2) is computed (R=basering).

Example:

```
LIB "homolog.lib";
ring r;
matrix n[2][3]=x,y,5,z,77,33;
print(kohom(n,3));
↦ x,0,0,y, 0, 0, 5, 0, 0,
↦ 0,x,0,0, y, 0, 0, 5, 0,
↦ 0,0,x,0, 0, y, 0, 0, 5,
↦ z,0,0,77,0, 0, 33,0, 0,
↦ 0,z,0,0, 77,0, 0, 33,0,
↦ 0,0,z,0, 0, 77,0, 0, 33
```

D.4.11.17 kontrahom

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: kontrahom(A,k); A=matrix, k=integer

Return: matrix Hom(A,R^k), i.e. let A be a matrix defining a map F1–>F2 of free R-modules, then the matrix of Hom(F2,R^k)–>Hom(F1,R^k) is computed (R=basering).

Example:

```
LIB "homolog.lib";
ring r;
matrix n[2][3]=x,y,5,z,77,33;
print(kontrahom(n,3));
↦ x,z, 0,0, 0,0,
↦ y,77,0,0, 0,0,
↦ 5,33,0,0, 0,0,
↦ 0,0, x,z, 0,0,
↦ 0,0, y,77,0,0,
↦ 0,0, 5,33,0,0,
↦ 0,0, 0,0, x,z,
↦ 0,0, 0,0, y,77,
↦ 0,0, 0,0, 5,33
```

D.4.11.18 KoszulHomology

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Compute: A presentation of the p-th Koszul homology module H_p(f_1,...,f_k;M’), where M’=coker(M) and f_1,...,f_k are the given (ordered list of non-zero) generators of the ideal I.
The computed presentation is minimized via prune.
In particular, if H_p(f_1,...,f_k;M’)=0 then the return value is 0.

Return: module H, s.th. coker(H) = H_p(f_1,...,f_k;M’).

Note: size of input ideal has to be <= 20.

Example:

```
LIB "homolog.lib";
ring R=0,x(1..3),dp;
ideal x=maxideal(1);
module M=0;
KoszulHomology(x,M,0);  // H_0(x,R), x=(x_1,x_2,x_3)
↦ _[1]=x(3)*gen(1)
↦ _[2]=x(2)*gen(1)
↦ _[3]=x(1)*gen(1)
KoszulHomology(x,M,1);  // H_1(x,R), x=(x_1,x_2,x_3)
↦ _[1]=0
qring S=std(x(1)*x(2));
module M=0;
ideal x=maxideal(1);
KoszulHomology(x,M,1);
↦ _[1]=-x(3)*gen(1)
↦ _[2]=-x(2)*gen(1)
↦ _[3]=-x(1)*gen(1)
KoszulHomology(x,M,2);
↦ _[1]=0
```

D.4.11.19 tensorMod

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Usage: tensorMod(M,N); M,N modules

Compute: presentation matrix A of the tensor product T of the modules M'=coker(M), N'=coker(N): if matrix(M) defines a map M: R^r–>R^s and matrix(N) defines a map N: R^p–>R^q, then A defines a presentation

```
R^(sp+rq) --A-> R^(sq)  --> T --> 0 .
```

Return: matrix A satisfying coker(A) = tensorprod(coker(M),coker(N)) .

Example:

```
LIB "homolog.lib";
ring A=0,(x,y,z),dp;
matrix M[3][3]=1,2,3,4,5,6,7,8,9;
matrix N[2][2]=x,y,0,z;
print(M);
↦ 1,2,3,
↦ 4,5,6,
↦ 7,8,9
print(N);
↦ x,y,
↦ 0,z
print(tensorMod(M,N));
↦ x,y,0,0,0,0,1,0,2,0,3,0,
↦ 0,z,0,0,0,0,0,1,0,2,0,3,
↦ 0,0,x,y,0,0,4,0,5,0,6,0,
↦ 0,0,0,z,0,0,0,4,0,5,0,6,
↦ 0,0,0,0,x,y,7,0,8,0,9,0,
↦ 0,0,0,0,0,z,0,7,0,8,0,9
```

D.4.11.20 Tor

Procedure from library `homolog.lib` (see Section D.4.11 [homolog_lib], page 1090).

Compute: a presentation of Tor_k(M',N'), for k=v[1],v[2],... , where M'=coker(M) and N'=coker(N): let

```
0 <-- M' <-- G0 <-M-- G1
0 <-- N' <-- F0 <--N- F1 <-- F2 <--...
```

be a presentation of M', resp. a free resolution of N', and consider the commutative diagram

```
           0                    0                  0
           |^                   |^                 |^
Tensor(M',Fk+1) -Ak+1-> Tensor(M',Fk) -Ak-> Tensor(M',Fk-1)
           |^                   |^                 |^
Tensor(G0,Fk+1) -Ak+1-> Tensor(G0,Fk) -Ak-> Tensor(G0,Fk-1)
                                |^                 |^
                                |C                 |B
                        Tensor(G1,Fk) ----> Tensor(G1,Fk-1)

       (Ak,Ak+1 induced by N and B,C induced by M).
```

Let K=modulo(Ak,B), J=module(C)+module(Ak+1) and Tor=modulo(K,J), then we have exact sequences

```
R^p  --K-> Tensor(G0,Fk) --Ak-> Tensor(G0,Fk-1)/im(B),

R^q -Tor-> R^p --K-> Tensor(G0,Fk)/(im(C)+im(Ak+1)).
```

Hence, Tor presents Tor_k(M',N').

Return: - if v is of type int: module Tor, a presentation of Tor_k(M',N');
- if v is of type intvec: a list of Tor_k(M',N') (k=v[1],v[2],...);
- in case of a third argument of any type: list l with

l[1] = module Tor/list of Tor_k(M',N'),
l[2] = SB of Tor/list of SB of Tor_k(M',N'),
l[3] = matrix/list of matrices, each representing a kbase of Tor_k(M',N')
(if finite dimensional), or 0.

Display: printlevel >=0: (affine) dimension of Tor_k for each k (default).
printlevel >=1: matrices Ak, Ak+1 and kbase of Tor_k in Tensor(G0,Fk) (if finite dimensional).

Note: In order to compute Tor_k(M,N) use the command Tor(k,syz(M),syz(N)); or: list P=mres(M,2); list Q=mres(N,2); Tor(k,P[2],Q[2]);

Example:

```
LIB "homolog.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 0,(x,y),dp;
ideal i    = x2,y;
ideal j    = x;
list E     = Tor(0..2,i,j);    // Tor_k(r/i,r/j) for k=0,1,2 over r
↦ // dimension of Tor_0:  0
↦ // vdim of Tor_0:       1
↦
```

```
↦ // Computing Tor_1 (help Tor; gives an explanation):
↦ // Let 0 <- coker(M) <- G0 <-M- G1 be the present. of coker(M),
↦ // and 0 <- coker(N) <- F0 <-N- F1 <- F2 <- ... a resolution of
↦ // coker(N), then Tensor(G0,F1)-->Tensor(G0,F0) is given by:
↦ x
↦ // and Tensor(G0,F2) + Tensor(G1,F1)-->Tensor(G0,F1) is given by:
↦ 0,x2,y
↦
↦ // dimension of Tor_1:  0
↦ // vdim of Tor_1:       1
↦
↦ // Computing Tor_2 (help Tor; gives an explanation):
↦ // Let 0 <- coker(M) <- G0 <-M- G1 be the present. of coker(M),
↦ // and 0 <- coker(N) <- F0 <-N- F1 <- F2 <- ... a resolution of
↦ // coker(N), then Tensor(G0,F2)-->Tensor(G0,F1) is given by:
↦ 0
↦ // and Tensor(G0,F3) + Tensor(G1,F2)-->Tensor(G0,F2) is given by:
↦ 1,x2,y
↦
↦ // dimension of Tor_2:  -1
↦
qring R    = std(i);
ideal j    = fetch(r,j);
module M   = [x,0],[0,x];
printlevel = 2;
module E1  = Tor(1,M,j);       // Tor_1(R^2/M,R/j) over R=r/i
↦ // Computing Tor_1 (help Tor; gives an explanation):
↦ // Let 0 <- coker(M) <- G0 <-M- G1 be the present. of coker(M),
↦ // and 0 <- coker(N) <- F0 <-N- F1 <- F2 <- ... a resolution of
↦ // coker(N), then Tensor(G0,F1)-->Tensor(G0,F0) is given by:
↦ x,0,
↦ 0,x
↦ // and Tensor(G0,F2) + Tensor(G1,F1)-->Tensor(G0,F1) is given by:
↦ x,0,x,0,
↦ 0,x,0,x
↦
↦ // dimension of Tor_1:  0
↦ // vdim of Tor_1:       2
↦
list l     = Tor(3,M,M,1);     // Tor_3(R^2/M,R^2/M) over R=r/i
↦ // Computing Tor_3 (help Tor; gives an explanation):
↦ // Let 0 <- coker(M) <- G0 <-M- G1 be the present. of coker(M),
↦ // and 0 <- coker(N) <- F0 <-N- F1 <- F2 <- ... a resolution of
↦ // coker(N), then Tensor(G0,F3)-->Tensor(G0,F2) is given by:
↦ x,0,0,0,
↦ 0,x,0,0,
↦ 0,0,x,0,
↦ 0,0,0,x
↦ // and Tensor(G0,F4) + Tensor(G1,F3)-->Tensor(G0,F3) is given by:
↦ x,0,0,0,x,0,0,0,
↦ 0,x,0,0,0,x,0,0,
↦ 0,0,x,0,0,0,x,0,
↦ 0,0,0,x,0,0,0,x
```

```
↦
↦ // dimension of Tor_3:  0
↦ // vdim of Tor_3:       4
↦
↦ // columns of matrix are kbase of Tor_3 in Tensor(G0,F3)
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,0,1,0,
↦ 0,0,0,1
↦
printlevel = p;
```

D.4.12 integralbasis_lib

Library: integralbasis.lib

Purpose: Integral basis in algebraic function fields

Authors: J. Boehm, boehm at mathematik.uni-kl.de
W. Decker, decker at mathematik.uni-kl.de
S. Laplagne, slaplagn at dm.uba.ar
G. Pfister, pfister at mathematik.uni-kl.de

Overview: Given an irreducible polynomial f in two variables defining a plane curve, this library implements algorithms for computing an integral basis of the integral closure of the affine coordinate ring in the algebraic function field.

Procedures:

D.4.12.1 integralBasis

Procedure from library `integralbasis.lib` (see Section D.4.12 [integralbasis_lib], page 1114).

Usage: integralBasis(f, intVar); f irreducible polynomial in two variables, intVar integer indicating that the intVar-th variable of the ring is the integral element.
The base ring must be a ring in two variables, and the polynomial f must be monic as polynomial in the intVar-th variable.
Optional parameters in list choose (can be entered in any order):
Algorithm:
- "normal" -> the integral basis is computed using the general normalization algorithm.
- "hensel" -> the integral bases is computed using an algorithm based on Puiseux expansions and Hensel lifting. (only available for polynomials with rational coefficients; default option in that case)
Options for normal algorithm:
- "global" -> computes the normalization of R / <f> and puts the results in integral basis shape.
- "local" -> computes the normalization at each component of the singular locus of R/<f> and puts everything together. (Default option for normal algorithm.)
Options for hensel algorithm:
- "opti" -> The integral basis for the branches are merged using a combinatorial approach. (Default option.)
- "noOpti" -> the integral basis for the different branches are merged using the Chinese remainder theorem. This is usually slower when the number of branches is large.

- "local" -> computes the normalization at each component of the singular locus of R/<f> and puts everything together. (Default option for normal algorithm.)
Other options:
- "modular" -> uses modular algorithms for computing Groebner bases, radicals and decompositions whenever possible. Can be used together with any of the other options. The ground field must have characteristic 0. (Default option for ground fields of characteristic 0.)
- "nonModular" -> do not uses modular algorithms. (Default option for ground fields of positive charecteristic.)
- "atOrigin" -> will compute the local contribution to the integral basis at the origin only (naturally, this contribution is only relevant if the curve defined by f has a singularity at the origin).
- "isIrred" -> assumes that the input polynomial f is irreducible, and therefore will not check this. If this option is given but f is not irreducible, the output might be wrong.
- list("inputJ", ideal inputJ) -> takes as initial test ideal the ideal inputJ. This option is only for use in other procedures. Using this option, the result might not be the integral basis. (When this option is given, the global option will be used.)
- list("inputC", ideal inputC) -> takes as initial conductor the ideal inputC. This option is only for use in other procedures. Using this option, the result might not be the integral basis. (When this option is given, the global option will be used.)
- "locBasis" -> when computing the integral basis at a prime or primary component, it computes a local basis, that is, a basis that is integral only over the ring localized at the component. This option is only valid when "atOrigin" is chosen or an initial test ideal or conductor is given.

Return: a list, say l, of size 2.
l[1] is an ideal I and l[2] is a polynomial D such that the integral basis is b_0 = I[1] / D, b_1 = I[2] / D, ..., b_{n-1} = I[n] / D.
That is, the integral closure of k[x] in the algebraic function field k(x,y) is
k[x] b_0 + k[x] b_1 + ... + k[x] b_{n-1},
where we assume that x is the transcendental variable, y is the integral element (indicated by intVar), f gives the integral equation and n is the degree of f as a polynomial in y.

Theory: We compute the integral basis of the integral closure of k[x] in k(x,y). When option "normal" is selected, the normalization of the affine ring k[x,y]/<f> is computed using procedure normal from normal.lib, which implements a general algorithm for normalization of rings by G. Greuel, S. Laplagne and F. Seelisch, and the k[x,y]-module generators are converted into a k[x]-basis.
When option "Hensel" is selected, the algorithm by J. Boehm, W. Decker, S. Laplagne and G. Pfister is used.

Example:

```
LIB "integralbasis.lib";
printlevel = printlevel+1;
ring s = 0,(x,y),dp;
poly f = y5-y4x+4y2x2-x4;
list l = integralBasis(f, 2);
↦ Computing the integral basis...
↦ DBG - optimize = 1
↦ --Computing the associated primes of the singular locus...
↦   (Using non-modular algorithm.)
↦ --Computing the integral basis at each component...
↦ ----Computing the integral basis of component
↦ 1
↦ ----Component:
↦ compo[1]=y
↦ compo[2]=x
↦ Integral basis computation finished.
l;
↦ [1]:
↦    _[1]=x3
↦    _[2]=21/524288x7-5/16384x6-1/128x5-1/2x4+x3y
↦    _[3]=-441/274877906944x10+777/4294967296x9-9/268435456x8-5/2048x6+1/25\
   6x5y-1/64x5+1/64x4y-1/4x4+x2y2
↦    _[4]=-441/274877906944x9y+777/4294967296x8y-9/268435456x7y-5/2048x5y+1\
   /256x4y2-1/64x4y+1/64x3y2-1/4x3y+xy3
↦    _[5]=567/2147483648x8y-21/134217728x7y2-63/67108864x8-51/67108864x7y+1\
   9/33554432x6y2-3/1048576x7-107/2097152x6y-5/1048576x5y2+21/524288x4y3+55/\
   131072x6-237/65536x5y+147/16384x4y2-69/16384x3y3+51/4096x5-21/512x4y+3/64\
   x3y2-3/128x2y3+3/32x4-3/8x3y+3/4x2y2-3/2xy3+y4-2x3+4x2y
↦ [2]:
↦    x3
// The integral basis of the integral closure of Q[x] in Q(x,y) consists
// of the elements of l[1] divided by the polynomial l[2].
printlevel = printlevel-1;
```

See also: Section D.4.23.1 [normal], page 1183.

D.4.12.2 polyDK

Procedure from library integralbasis.lib (see Section D.4.12 [integralbasis_lib], page 1114).

Usage: polyDK(d,k,#); d integer, k<=d integer.

Return: polynomial of degree d in the second variable of the ring with an ordinary multiple point at the origin of order k.

Example:

```
LIB "integralbasis.lib";
ring R = 0, (x,y), dp;
int k = 3;
int d = 6;
// Polynomial of degree 6 in y with an ordinary multiple point
// at the origin of order k.
poly f = polyDK(d, k, 1231);
f;
↦ -1/3x6-4/3x5y+x4y2+1/3x3y3-x2y4-4/3xy5+y6+1/3x5-x4y+x3y2+1/3x2y3-1/3xy4+4\
   /3y5+x4+4/3x3y-2/3x2y2+2/3xy3-y4+x3+2/3xy2+4/3y3
// The integral basis of R / <f>
list l = integralBasis(f, 2, "atOrigin");
l;
↦ [1]:
↦    _[1]=x2
↦     _[2]=x2y
↦     _[3]=x2y2
↦     _[4]=-1413399/32768x8y2-1016807/12288x8y+19063/2048x7y2+1386535/32768x\
   8+15505/1536x7y-657/1024x6y2-151351/6144x7+2399/384x6y-65/32x5y2+36083/30\
   72x6-155/24x5y+15/16x4y2-191/96x5+7/6x4y-11/6x3y2+x2y3-5/48x4-x3y+4/3x2y2\
   +7/6x3-x2y+4/3x2
↦     _[5]=-1413399/32768x7y3-1016807/12288x7y2+19063/2048x6y3+1386535/32768\
   x7y+15505/1536x6y2-657/1024x5y3-151351/6144x6y+2399/384x5y2-65/32x4y3+360\
   83/3072x5y-155/24x4y2+15/16x3y3-191/96x4y+7/6x3y2-11/6x2y3+xy4-5/48x3y-x2\
   y2+4/3xy3+7/6x2y-xy2+4/3xy
↦     _[6]=-1413399/32768x6y4-1016807/12288x6y3+19063/2048x5y4+1386535/32768\
   x6y2+15505/1536x5y3-657/1024x4y4-151351/6144x5y2+2399/384x4y3-65/32x3y4+3\
   6083/3072x4y2-155/24x3y3+15/16x2y4-191/96x3y2+7/6x2y3-11/6xy4+y5-5/48x2y2\
   -xy3+4/3y4+7/6xy2-y3+4/3y2
↦  [2]:
↦     x2
```

D.4.12.3 monic

Procedure from library `integralbasis.lib` (see Section D.4.12 [integralbasis_lib], page 1114).

Usage: monic(f); f polynomial in two variables whose leadi.

Return: a multiple of f monic as polynomial in the second variables, polynomial of degree d in the second variable of the ring with an ordinary multiple point at the origin of order k.

Assume: f is a bivariate polynomial whose leading coefficients as polynomial in the second variable is a unit.

Example:

```
LIB "integralbasis.lib";
ring R = 0, (x,y), dp;
poly f = -3y3 + 3x2y + y2 + 1;
monic(f);
↦  -x2y+y3-1/3y2-1/3
```

D.4.12.4 henselGlobal

Procedure from library `integralbasis.lib` (see Section D.4.12 [integralbasis_lib], page 1114).

Usage: henselGlobal(f,g,h,order); h is polynomial in x and y. f and g are polynomials in y such that f(y)*g(y) = h(0,y) and <f, g> = 1.

Return: polynomials f1 and g1 such that
1) h = f1*g1 up to the required order in x.
2) f1(0,y) = f, g1(0,y) = g

Example:

```
LIB "integralbasis.lib";
ring R = 0, (x,y), dp;
// Polynomial of degree 6 in y with an ordinary multiple point
// at the origin of order k.
poly h = (y2 + 3xy + x3 + x4)*(y3 + 2x + 1);
poly f = y2;
poly g = y3 + 1;
henselGlobal(f, g, h, 3);
```

```
↦ [1]:
↦    x3+3xy+y2
↦ [2]:
↦    y3+2x+1
```

D.4.13 intprog_lib

Library: intprog.lib

Purpose: Integer Programming with Groebner Basis Methods

Author: Christine Theis, email: ctheis@math.uni-sb.de

Procedures:

D.4.13.1 solve_IP

Procedure from library `intprog.lib` (see Section D.4.13 [intprog_lib], page 1118).

Usage: solve_IP(A,bx,c,alg); A intmat, bx intvec, c intvec, alg string.
solve_IP(A,bx,c,alg); A intmat, bx list of intvec, c intvec, alg string.
solve_IP(A,bx,c,alg,prsv); A intmat, bx intvec, c intvec, alg string, prsv intvec.
solve_IP(A,bx,c,alg,prsv); A intmat, bx list of intvec, c intvec, alg string, prsv intvec.

Return: same type as bx: solution of the associated integer programming problem(s) as explained in

Section C.6 [Toric ideals and integer programming], page 781.

Note: This procedure returns the solution(s) of the given IP-problem(s) or the message 'not solvable'.
One may call the procedure with several different algorithms:
- the algorithm of Conti/Traverso (ct),
- the positive variant of the algorithm of Conti/Traverso (pct),
- the algorithm of Conti/Traverso using elimination (ect),
- the algorithm of Pottier (pt),
- an algorithm of Bigatti/La Scala/Robbiano (blr),
- the algorithm of Hosten/Sturmfels (hs),
- the algorithm of DiBiase/Urbanke (du).
The argument 'alg' should be the abbreviation for an algorithm as above: ct, pct, ect, pt, blr, hs or du.

'ct' allows computation of an optimal solution of the IP-problem directly from the right-hand vector b.
The same is true for its 'positive' variant 'pct' which may only be applied if A and b have nonnegative entries.
All other algorithms need initial solutions of the IP-problem.

If 'alg' is chosen to be 'ct' or 'pct', bx is read as the right hand vector b of the system Ax=b. b should then be an intvec of size m where m is the number of rows of A. Furthermore, bx and A should be nonnegative if 'pct' is used. If 'alg' is chosen to be 'ect','pt','blr','hs' or 'du', bx is read as an initial solution x of the system Ax=b. bx should then be a nonnegative intvec of size n where n is the number of columns of A.

If 'alg' is chosen to be 'blr' or 'hs', the algorithm needs a vector with positive coefficients in the row space of A.
If no row of A contains only positive entries, one has to use the versions of solve_IP which take such a vector prsv as an argument.

solve_IP may also be called with a list bx of intvecs instead of a single intvec.

Example:

```
LIB "intprog.lib";
// 1. call with single right-hand vector
intmat A[2][3]=1,1,0,0,1,1;
intvec b1=1,1;
intvec c=2,2,1;
intvec solution_vector=solve_IP(A,b1,c,"pct");
solution_vector;"";
↦ 0,1,0
↦
// 2. call with list of right-hand vectors
intvec b2=-1,1;
list l=b1,b2;
l;
↦ [1]:
↦    1,1
↦ [2]:
↦    -1,1
list solution_list=solve_IP(A,l,c,"ct");
solution_list;"";
↦ [1]:
↦    0,1,0
↦ [2]:
↦    not solvable
↦
// 3. call with single initial solution vector
A=2,1,-1,-1,1,2;
b1=3,4,5;
solve_IP(A,b1,c,"du");"";
↦ 0,7,2
↦
// 4. call with single initial solution vector
//    and algorithm needing a positive row space vector
solution_vector=solve_IP(A,b1,c,"hs");"";
↦ ERROR: The chosen algorithm needs a positive vector in the row space of t\
   he matrix.
↦ 0
↦
// 5. call with single initial solution vector
//     and positive row space vector
intvec prsv=1,2,1;
solution_vector=solve_IP(A,b1,c,"hs",prsv);
solution_vector;"";
↦ 0,7,2
↦
// 6. call with list of initial solution vectors
//    and positive row space vector
b2=7,8,0;
l=b1,b2;
l;
↦ [1]:
```

```
↦    3,4,5
↦ [2]:
↦    7,8,0
solution_list=solve_IP(A,l,c,"blr",prsv);
solution_list;
↦ [1]:
↦    0,7,2
↦ [2]:
↦    7,8,0
```

See also: Section C.6.4 [Integer programming], page 783; Section D.4.13 [intprog_lib], page 1118; Section D.4.35 [toric_lib], page 1387.

D.4.14 moddiq_lib

Library: moddiq.lib

Purpose: Double ideal quotient using modular methods

Authors: Y. Ishihara yishihara@rikkyo.ac.jp

Overview: A library for computing ideal quotient and saturation in the polynomial ring over the rational numbers using modular methods.

References:

M. Noro, K. Yokoyama: Usage of Modular Techniques for Efficient Computation of Ideal Operations. Math.Comput.Sci. 12: 1, 1-32. (2017).

Procedures:

D.4.14.1 modQuotient

Procedure from library `moddiq.lib` (see Section D.4.14 [moddiq_lib], page 1120).

Usage: modQuotient(I,J); I,J ideal

Return: a standard basis of (I:J)

Note: The procedure computes a standard basis of (I:J) (over the rational numbers) by using modular methods.

Example:

```
LIB "moddiq.lib";
ring r=0,x(1..6),dp;
ideal i=cyclic(6);
ideal j=-15*var(5)+16*var(6)^3-60*var(6)^2+225*var(6)-4,2*var(5)^2-7*var(5)+2*var(6)
modQuotient(i,modQuotient(i,j));
↦ _[1]=x(4)+1/4*x(5)+1/4*x(6)
↦ _[2]=x(3)+1/4*x(5)+1/4*x(6)
↦ _[3]=x(2)+1/4*x(5)+1/4*x(6)
↦ _[4]=x(1)+1/4*x(5)+1/4*x(6)
↦ _[5]=x(5)*x(6)-1/4*x(5)-1/4*x(6)+1
↦ _[6]=x(5)^2+x(6)^2-7/2*x(5)-7/2*x(6)+14
↦ _[7]=x(6)^3-15/4*x(6)^2-15/16*x(5)+225/16*x(6)-1/4
ideal id2=x(1)^2+x(1)*x(2)*x(3),x(2)^2-x(3)^3*x(2),x(3)^3+x(2)^5*x(1)*x(3);
quotient(id2,maxideal(3));
↦ _[1]=x(1)*x(2)*x(3)+x(1)^2
↦ _[2]=x(2)*x(3)^3-x(2)^2
↦ _[3]=x(1)*x(3)^3+x(2)^2*x(3)
↦ _[4]=x(1)^2*x(3)^2+x(1)*x(2)^2
↦ _[5]=x(2)^3*x(3)+x(1)*x(2)^2
↦ _[6]=x(2)^4+x(1)^3
↦ _[7]=x(1)*x(2)^3-x(1)^3*x(3)
↦ _[8]=x(1)^5+x(3)^3
↦ _[9]=x(3)^6-x(2)^2
↦ _[10]=x(1)^4*x(2)^2-x(3)^5
```

See also: Section D.2.6.1 [modular], page 883; Section 5.1.127 [quotient], page 248.

D.4.14.2 modSat

Procedure from library `moddiq.lib` (see Section D.4.14 [moddiq_lib], page 1120).

Usage: modSat(I,J); I,J ideal

Return: a standard basis of (I:J^\infty)

Note: The procedure computes a standard basis of (I:J^\infty) (over the rational numbers) by using modular methods.

Example:

```
LIB "moddiq.lib";
ring r=0,x(1..6),dp;
ideal i=cyclic(6);
ideal j=-15*var(5)+16*var(6)^3-60*var(6)^2+225*var(6)-4,2*var(5)^2-7*var(5)+2*var(6)
modSat(i,modSat(i,j)[1])[1];
↦ _[1]=x(4)+1/4*x(5)+1/4*x(6)
↦ _[2]=x(3)+1/4*x(5)+1/4*x(6)
↦ _[3]=x(2)+1/4*x(5)+1/4*x(6)
↦ _[4]=x(1)+1/4*x(5)+1/4*x(6)
↦ _[5]=x(5)*x(6)-1/4*x(5)-1/4*x(6)+1
↦ _[6]=x(5)^2+x(6)^2-7/2*x(5)-7/2*x(6)+14
↦ _[7]=x(6)^3-15/4*x(6)^2-15/16*x(5)+225/16*x(6)-1/4
poly F     = x(1)^5+x(2)^5+(x(1)-x(2))^2*x(1)*x(2)*x(3);
ideal J    = jacob(F);
modSat(J,maxideal(1));
↦ [1]:
↦    _[1]=x(2)^4+1/5*x(1)^3*x(3)-4/5*x(1)^2*x(2)*x(3)+3/5*x(1)*x(2)^2*x(3)
↦    _[2]=x(1)^3*x(2)-2*x(1)^2*x(2)^2+x(1)*x(2)^3
↦    _[3]=x(1)^4+3/5*x(1)^2*x(2)*x(3)-4/5*x(1)*x(2)^2*x(3)+1/5*x(2)^3*x(3)
↦    _[4]=x(1)^2*x(2)^3-7/15*x(1)^2*x(2)^2*x(3)+4/15*x(1)*x(2)^3*x(3)-1/75*\
   x(1)^3*x(3)^2-2/75*x(1)^2*x(2)*x(3)^2+1/15*x(1)*x(2)^2*x(3)^2-2/75*x(2)^3\
   *x(3)^2
↦    _[5]=x(1)*x(2)^3*x(3)^2+29/220*x(1)^3*x(3)^3-71/220*x(1)^2*x(2)*x(3)^3\
   +27/220*x(1)*x(2)^2*x(3)^3+3/44*x(2)^3*x(3)^3
↦    _[6]=x(1)^2*x(2)^2*x(3)^2-44/41*x(1)*x(2)^3*x(3)^2-17/410*x(1)^3*x(3)^\
   3+101/410*x(1)^2*x(2)*x(3)^3-19/82*x(1)*x(2)^2*x(3)^3+11/410*x(2)^3*x(3)^\
   3
↦    _[7]=x(1)^3*x(3)^4-7*x(1)^2*x(2)*x(3)^4+7*x(1)*x(2)^2*x(3)^4-x(2)^3*x(\
   3)^4
↦ [2]:
↦ 0
```

See also: Section D.2.6.1 [modular], page 883; Section D.4.7.8 [sat_with_exp], page 1065.

D.4.15 modules_lib

Library: modules.lib

Purpose: Modules

Authors: J. Boehm, boehm@mathematik.uni-kl.de
D. Wienholz wienholz@mathematik.uni-kl.de
C. Koenen koenen@rhrk.uni-kl.de
M. Mayer mayer@mathematik.uni-kl.de

Overview: This library is used for the computation of graded free resolutions with an own graduation of the monomials. For these Resolution is a new class of modules needed. These modules, can be computed via the image, kernel, cokernel of a matrix or the subquotient of two matrices. The used matrices also have a free module as source and target, with graded generators if the matrix is homogeneous. A matrix of this new form is created by a normal matrix, source, target and the graduatin, if the matrix is homogeneous, are done automatically. With this matrices it is then possible to compute the new class of modules.
This library also offers the opppurtunity to create R-module-homomorphisms betweens two modules. For these homorphisms the kernel can be computed an will be returned as a module of the new class.

This is experimental work in progress!!!

Types: Matrix the class of matrices with source and target in form of free modules FreeModule free modules representet with the ring and degree Resolution class of graded resolutions
Module modules represented by either the image, coker, kernelof a matrix or the subquotient of two matrices Vector element of a Module
Ideal same as ideal, but with it's own basering saved, used to compute resolutions Homomorphism class of R-module-homomormphisms

Procedures:

D.4.15.1 id

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: id(n); n integer

Return: returns the n x n identity matrix, with nongraded free modules for source and target

Example:

```
LIB "modules.lib";
ring r;
int n=4;
id(n);
↦     {0} {0} {0} {0}
↦ {0}   1   0   0   0
↦ {0}   0   1   0   0
↦ {0}   0   0   1   0
↦ {0}   0   0   0   1
↦
```

D.4.15.2 zero

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: zero(n); n and m integer

Return: returns the n x m zero matrix, with nongraded free modules for source and target

Example:

```
LIB "modules.lib";
ring r;
int n=4;
int m=3;
zero(n,m);
↦     {0} {0} {0}
↦ {0}   0   0   0
↦ {0}   0   0   0
↦ {0}   0   0   0
↦ {0}   0   0   0
↦
```

D.4.15.3 freeModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: freeModule(r,n,l); r ring, n integer, l list

Return: a free Module over the ring r, with rank n, and degrees l for the generators

Note: -1 for nor graduation and 0 to set every degree to 0

Example:

```
LIB "modules.lib";
ring r;
int n=3;
list l=1,2,3;
freeModule(r,n,l);
↦  3
↦ r
↦ free Module
↦ Degrees of the generators: {1} {2} {3}
↦
```

D.4.15.4 makeMatrix

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: makeMatrix(m), m matrix
RETURN Matrix, with graded source and target if the matrix is homogeneous

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y3,z,xz;
m;
↦ m[1,1]=x
↦ m[1,2]=y3
↦ m[2,1]=z
↦ m[2,2]=xz
Matrix M=m;
```

D.4.15.5 makeIdeal

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: makeIdeal(i) or Ideal I=i; i ideal

Return: Ideal with saved basering

Example:

```
LIB "modules.lib";
ring r;
ideal i=x,y,z+x;
i;
↦ i[1]=x
↦ i[2]=y
↦ i[3]=x+z
Ideal I=i;
```

D.4.15.6 Target

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Target(M); M Matrix

Return: FreeModule, target of the Matrix

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y3,z,xz;
Matrix M=m;
M;
↦ x,y3,
↦ z,xz
↦
Target(M);
↦  2
↦ r
↦ free Module
↦
```

D.4.15.7 Source

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Source(M); M Matrix

Return: FreeModule, source of the Matrix

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y3,z,xz;
```

```
Matrix M=m;
M;
↦ x,y3,
↦ z,xz
↦
Source(M);
↦  2
↦ r
↦ free Module
↦
```

D.4.15.8 printMatrix

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: printMatrix(M); or M; M Matrix

Return: nothing, prints the matrix with degrees of the generators from target and source

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y3,z,xz;
Matrix M=m;
M;
↦ x,y3,
↦ z,xz
↦
```

D.4.15.9 printFreeModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: printFreeModule(M); or M; M FreeModule

Return: nothing, prints a free Module

Example:

```
LIB "modules.lib";
ring r;
int n=3;
list l=1,2,3;
freeModule(r,n,l);
↦  3
↦ r
↦ free Module
↦ Degrees of the generators: {1} {2} {3}
↦
```

D.4.15.10 printResolution

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: printResolution(R); or R; R Resolution

Return: nothing, prints the resolution

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
N;
↦ cokernel | x y z |
↦
↦
Resolution R = Res(N);
R;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
↦
```

D.4.15.11 printModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: nothing, prints the Module

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
Matrix M=m;
M;
↦     {1}  {2}
↦ {0}   x   y2
↦ {0}   z   xz
↦
Module N=image(M);
N;
↦ image | x y2 |
↦       | z xz |
↦
↦
```

D.4.15.12 printHom

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: printHom(M); f a Homomorphism

Return: nothing, prints f

Example:

```
LIB "modules.lib";
ring R=0,(x,y),(lp,c);
Matrix M=id(2);
```

```
Module src=image(M);
matrix rules[2][2]=x,y,x2,y2;
Module tar=coker(M);
src;
↦ R^2
↦
↦
tar;
↦ cokernel | 1 0 |
↦          | 0 1 |
↦
↦
rules;
↦ rules[1,1]=x
↦ rules[1,2]=y
↦ rules[2,1]=x2
↦ rules[2,2]=y2
Homomorphism f=homomorphism(rules,src,tar);
f;
↦ | x  y  |
↦ | x2 y2 |
↦
↦ cokernel | 1 0 | <--- R^2
↦          | 0 1 |
↦
```

D.4.15.13 mRes

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: mRes(M,n); M Module or Ideal, n integer

Return: Resolution, minimized resolution with graded modules

Note: n is optional, if n ist positiv only that many steps will be computed use R.dd[i]; to return the different modules as image of matrices, R Resolution i integer EXAMPLE. example mRes, shows an example

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
N;
↦ cokernel | x y z |
↦
↦
Resolution R = mRes(N);
R;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦
```

```
↦
R.dd[2];
↦       {2}  {2}  {2}
↦ {1}     y    x    0
↦ {1}    -z    0    x
↦ {1}     0   -z   -y
↦
```

D.4.15.14 sRes

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: sRes(M,n); M Module or Ideal, n integer

Return: Resolution, with graded modules, computed with Schreyer's method using the function sres

Note: n is optional, if n ist positiv only that many steps will be computed use R.dd[i]; to return the different modules as image of matrices, R Resolution i integer EXAMPLE. example sRes, shows an example

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
N;
↦ cokernel | x y z |
↦
↦
Resolution R = sRes(N);
R;
↦  1      3      3      1
↦ r <--   r <--   r <--   r
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
↦
R.dd[2];
↦       {2}  {2}  {2}
↦ {1}    -y   -x    0
↦ {1}     z    0   -x
↦ {1}     0    z    y
↦
```

D.4.15.15 Res

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Res(M,n); M Module or Ideal, n integer

Return: Resolution, resolution with graded modules

Note: n is optional, if n ist positiv only that many steps will be computed use R.dd[i]; to return the different modules as image of matrices, R Resolution i integer EXAMPLE. example Res, shows an example

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
N;
↦ cokernel | x y z |
↦
↦
Resolution R = Res(N);
R;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦ resolution not minimized yet
↦
↦
R.dd[2];
↦      {2}  {2}  {2}
↦ {1}   -y   -x    0
↦ {1}    z    0   -x
↦ {1}    0    z    y
↦
```

D.4.15.16 Betti

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Betti(reso); reso Resolution

Return: intmat, Bettimatrix of the resolution

Note: for a clear overview use printBetti

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
Resolution R=mRes(N);
R;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦
↦
Betti(R);
↦ 1,3,3,1
```

D.4.15.17 printBetti

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: printBetti(resi), resi Resolution

Return: nothing, prints the Bettimatrix of the Resolution

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y,z;
Matrix M=m;
Module N=coker(M);
Resolution R=mRes(N);
R;
↦  1      3      3      1
↦ r <--  r <--  r <--  r
↦
↦ 0      1      2      3
↦
↦
Betti(R);
↦ 1,3,3,1
```

D.4.15.18 SetDeg

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: SetDeg(l), l list or intvec

Return: nothing, saves own degrees for the different variables of the basering

Note: should be used after decleration of the ring and shouldn't be changed afterwards

Example:

```
LIB "modules.lib";
ring r;
Deg(x);
↦ 1
list l=2,2,2;
SetDeg(l);
Deg(x);
↦ 2
```

D.4.15.19 Deg

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: same as deg

Note: only needed if custom degrees are set with SetDeg

Example:

```
LIB "modules.lib";
ring r;
Deg(x);
```

```
↦ 1
list l=2,2,2;
SetDeg(l);
Deg(x);
↦ 2
```

D.4.15.20 Degree

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Degree(M); M FreeModule

Return: list, degrees of the generators from the module, if they are graded

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y3,z,xz;
Matrix Ma=m;
FreeModule M=Source(Ma);
M;
↦  2
↦ r
↦ free Module
↦
Degree(M);
↦ The module isn't graded
```

D.4.15.21 Degrees

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Degrees(M); M a Module

Return: list, grading of the Module

Example:

```
LIB "modules.lib";
ring r;
matrix ma[2][2]=x,y,x,y;
Matrix m=ma;
Module M=image(m);
M;
↦ image | x y |
↦       | x y |
↦
↦
Degrees(M);
↦ [1]:
↦    1
↦ [2]:
↦    1
```

D.4.15.22 subquotient

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: subquotient(gens,rels); gens and rels Matrices

Return: Module, the subquotient of two entered Matrices

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
Matrix M=m;
matrix n[2][3]=z2,xyz,x2y2,xy,x3,y4;
Matrix N=n;
M;
↦      {1}  {2}
↦ {0}    x    y2
↦ {0}    z    xz
↦
N;
↦       {2}   {3}    {4}
↦ {0}    z2    xyz    x2y2
↦ {0}    xy     x3      y4
↦
subquotient(M,N);
↦ subquotient (| x y2 |, | z2 xyz x2y2 |)
↦               | z xz |  | xy x3  y4   |
↦
↦
```

D.4.15.23 coker

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: coker(M), M a Matrix

Return: Module, the coker of M

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
Matrix M=m;
M;
↦      {1}  {2}
↦ {0}    x    y2
↦ {0}    z    xz
↦
coker(M);
↦ cokernel | x y2 |
↦          | z xz |
↦
↦
```

D.4.15.24 image

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: image(M); M a Matrix

Return: Module, the image of M

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
Matrix M=m;
M;
↦     {1}  {2}
↦ {0}   x   y2
↦ {0}   z   xz
↦
image(M);
↦ image | x y2 |
↦       | z xz |
↦
↦
```

D.4.15.25 Ker

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: Ker(M); M a Matrix

Return: Module, image of a Matrix N with image(N)=Ker(M)

Example:

```
LIB "modules.lib";
ring r;
matrix m[1][3]=x,y2,z3;
Matrix M=m;
M;
↦     {1}  {2}  {3}
↦ {0}   x   y2   z3
↦
Ker(M);
↦ image | y2 z3 0   |
↦       | -x 0  z3  |
↦       | 0  -x -y2 |
↦
↦
```

D.4.15.26 compareModules

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: compareModules(M,N); or M==N; compares two Modules up to isomorphism

Return: 1 or 0, if the are ismomophic or aren't

Example:

```
LIB "modules.lib";
ring r;
matrix ma[2][2]=x,y,x,y;
Matrix m=ma;
Module M=image(m);
matrix na[2][1]=-y,x;
Matrix n=na;
M;
↦ image | x y |
↦       | x y |
↦
↦
Module N=image(n);
N;
↦ image | -y |
↦       | x  |
↦
↦
N==M;
↦ 0
N=coker(n);
N;
↦ cokernel | -y |
↦          | x  |
↦
↦
N==M;
↦ 0
```

D.4.15.27 addModules

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: addModules(M,N); or M+N; M and N Modules

Return: Module, sum of the two Modules

Example:

```
LIB "modules.lib";
ring r;
matrix ma[2][2]=x,y,x2,y2;
Matrix m=ma;
Module M=image(m);
matrix na[2][2]=xy,x2,y2,x;
Matrix n=na;
Module N=image(na);
M;
↦ image | x  y  |
↦       | x2 y2 |
↦
↦
N;
↦ image | xy x2 |
↦       | y2 x  |
↦
↦
N+M;
↦ image | xy x2 x  y  |
↦       | y2 x  x2 y2 |
↦
↦
```

D.4.15.28 homomorphism

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: homomorphism(rules,sources,targets); rules a matrix, sources and targets are Modules Return: Homomorphism

Example:

```
LIB "modules.lib";
ring R=0,(x,y),(lp,c);
Matrix M=id(2);
Module src=image(M);
matrix rules[2][2]=x,y,x2,y2;
Module tar=coker(M);
src;
↦ R^2
↦
↦
tar;
↦ cokernel | 1 0 |
↦          | 0 1 |
↦
↦
rules;
↦ rules[1,1]=x
↦ rules[1,2]=y
↦ rules[2,1]=x2
↦ rules[2,2]=y2
homomorphism(rules,src,tar);
↦ | x  y  |
↦ | x2 y2 |
↦
↦ cokernel | 1 0 | <--- R^2
↦          | 0 1 |
↦
```

D.4.15.29 target

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: target(f); f Homomorphism

Return: Module, the target of f

Example:

```
LIB "modules.lib";
ring R=0,(x,y),(lp,c);
```

```
Matrix M=id(2);
Module src=image(M);
matrix rules[2][2]=x,y,x2,y2;
Module tar=coker(M);
src;
↦ R^2
↦
↦
tar;
↦ cokernel | 1 0 |
↦          | 0 1 |
↦
↦
rules;
↦ rules[1,1]=x
↦ rules[1,2]=y
↦ rules[2,1]=x2
↦ rules[2,2]=y2
Homomorphism f=homomorphism(rules,src,tar);
f;
↦ | x  y  |
↦ | x2 y2 |
↦
↦ cokernel | 1 0 | <--- R^2
↦          | 0 1 |
↦
target(f);
↦ cokernel | 1 0 |
↦          | 0 1 |
↦
↦
```

D.4.15.30 source

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: source(f); f Homomorphism

Return: Module, the source of f

Example:

```
LIB "modules.lib";
ring R=0,(x,y),(lp,c);
Matrix M=id(2);
Module src=image(M);
matrix rules[2][2]=x,y,x2,y2;
Module tar=coker(M);
src;
↦ R^2
↦
↦
tar;
↦ cokernel | 1 0 |
↦          | 0 1 |
↦
↦
rules;
↦ rules[1,1]=x
↦ rules[1,2]=y
↦ rules[2,1]=x2
↦ rules[2,2]=y2
Homomorphism f=homomorphism(rules,src,tar);
f;
↦ | x  y  |
↦ | x2 y2 |
↦
↦ cokernel | 1 0 | <--- R^2
↦          | 0 1 |
↦
source(f);
↦ R^2
↦
↦
```

D.4.15.31 compareMatrix

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: compareMatrix(M,N); or M==N; compares two Matrices

Return: 1 or 0, if the are the same or aren't

Example:

```
LIB "modules.lib";
ring r;
matrix ma[2][2]=x,y,x,y;
Matrix M=ma;
matrix na[2][1]=-y,x;
Matrix N=na;
M;
↦      {1} {1}
↦ {0}   x   y
↦ {0}   x   y
↦
N;
↦       {1}
↦ {0}   -y
↦ {0}    x
↦
N==M;
↦ 0
M==M;
↦ 1
```

D.4.15.32 freeModule2Module

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: freeModule2Module(F); F FreeModule

Return: returns F as a Module

Example:

```
LIB "modules.lib";
ring r;
list L = 1,1,1;
FreeModule F = freeModule(r,3,L);
freeModule2Module(F);
↦ R^3
↦
↦
```

D.4.15.33 makeVector

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: a Vector V, element of M with entries v

Example:

```
LIB "modules.lib";
ring r;
Module M = image(id(3));
makeVector([x,y,z],M);
↦ | x |
↦ | y |
↦ | z |
↦
↦
```

D.4.15.34 netVector

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: netVector(V); V Vector

Return: pretty print for Vector

Example:

```
LIB "modules.lib";
ring r;
Module M = image(id(3));
Vector V = makeVector([x,y,z],M);
netVector(V);
↦ | x |
↦ | y |
↦ | z |
↦
```

D.4.15.35 netMatrix

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: nothing, prints M

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
Matrix M=m;
netMatrix(M);
↦ | x y2 |
↦ | z xz |
↦
```

D.4.15.36 presentation

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Subquotient M converted to coker(C)

Example:

```
LIB "modules.lib";
ring R = 0,(x,y),dp;
matrix a[1][2] = x,y;
Matrix A = a;
matrix b[1][2] = x2,y2;
Matrix B = b;
Module M = subquotient(A,B);
presentation(M);
↦ cokernel | 0 x -y |
↦          | y 0 x  |
↦
↦
```

D.4.15.37 tensorMatrix

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Tensorprodukt of A,B

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][2]=x,y2,z,xz;
matrix n[2][2]=1,2,3,4;
Matrix M = m;
Matrix N = n;
tensorMatrix(M,N);
↦       {1}  {1}   {2}   {2}
↦ {0}     x   2x    y2   2y2
↦ {0}    3x   4x   3y2   4y2
↦ {0}     z   2z    xz   2xz
↦ {0}    3z   4z   3xz   4xz
↦
```

D.4.15.38 tensorModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Tensorprodukt of M,N

Example:

```
LIB "modules.lib";
ring R = 0,(x,y,z),dp;
matrix a[1][2] = x,y;
Matrix A = a;
matrix b[1][2] = x2,y2;
Matrix B = b;
Module M = subquotient(A,B);
M;
↦ subquotient (| x y |, | x2 y2 |)
↦
↦
matrix c[2][2]=x,y2,z,xz;
Matrix C=c;
matrix d[2][3]=z2,xyz,x2y2,xy,x3,y4;
Matrix D=d;
Module N = subquotient(C,D);
N;
↦ subquotient (| x y2 |, | z2 xyz x2y2 |)
↦              | z xz |  | xy x3  y4   |
↦
↦
tensorModule(M,N);
↦ cokernel | 0 x -y 0 0 0  -xyz+y2 0       x2z-xyz 0       y4-xyz2-x2z+xyz+\
  y2z-y2 0                      -xy2z-y3z+xy2+y3+x2z-xyz+xz2+y2 0          \
                       -y3z+xz2 0        xy2z2-xy2z-y3z+y3 0               \
    y4z-y4-x2z2+xyz2+x2z-xyz-y2z+y2 0                               xy3z-xy\
  3+xy2z+y3z-xy2-y3-x2z+xyz-xz2-y2 0                                       \
  0       0       x2y2z-x2y2+xy2z+y3z-xy2-y3-x2z+xyz-xz2-y2 0              \
                             |
↦          | y 0 x  0 0 0  0       -xyz+y2 0       x2z-xyz 0               \
         y4-xyz2-x2z+xyz+y2z-y2 0                               -xy2z-y3z+x\
  y2+y3+x2z-xyz+xz2+y2 0        -y3z+xz2 0                 xy2z2-xy2z-y3z+y\
  3 0                               y4z-y4-x2z2+xyz2+x2z-xyz-y2z+y2 0      \
                                   xy3z-xy3+xy2z+y3z-xy2-y3-x2z+xyz-xz2-y2 \
  0       0       0                                         x2y2z-x2y2+xy2z\
  +y3z-xy2-y3-x2z+xyz-xz2-y2 |
↦          | 0 0 0  0 x -y z2-x    0       0       0       -xy2+z3-z2+x    \
         0                      -y2z+yz2-x2-xy-xz+z2-x          0          \
                       0        0        -yz3+x2z+yz2-xy   0               \
    xy2-z3+z2-x                     0                               -y2z2+x\
  2y+y2z-yz2+x2+xy+xz-z2+x         0                                       \
  xyz-y2z 0       -xyz2+x3+y2z-yz2+x2+xy+xz-z2+x            0              \
                             |
↦          | 0 0 0  y 0 x  0       z2-x    0       0       0               \
         -xy2+z3-z2+x           0                               -y2z+yz2-x2\
  -xy-xz+z2-x          0        0        0                 -yz3+x2z+yz2-xy \
    0                               xy2-z3+z2-x                     0      \
                                   -y2z2+x2y+y2z-yz2+x2+xy+xz-z2+x         \
  0       xyz-y2z 0                                         -xyz2+x3+y2z-yz\
  2+x2+xy+xz-z2+x            |
↦
↦
```

D.4.15.39 tensorModFreemod

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Tensorprodukt of M,F

Example:

```
LIB "modules.lib";
ring R = 0,(x,y,z),dp;
matrix a[1][2] = x,y;
Matrix A = a;
matrix b[1][2] = x2,y2;
Matrix B = b;
Module M = subquotient(A,B);
M;
↦ subquotient (| x y |, | x2 y2 |)
↦
↦
FreeModule F = freeModule(R,3,0);
F;
↦  3
↦ R
↦ free Module
↦ Degrees of the generators: {0} {0} {0}
↦
tensorModFreemod(M,F);
↦ cokernel | 0 0 0 x 0 0 -y 0  0  |
↦          | 0 0 0 0 x 0 0  -y 0  |
↦          | 0 0 0 0 0 x 0  0  -y |
↦          | y 0 0 0 0 0 x  0  0  |
↦          | 0 y 0 0 0 0 0  x  0  |
↦          | 0 0 y 0 0 0 0  0  x  |
↦
↦
```

D.4.15.40 tensorFreemodMod

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

D.4.15.41 tensorFreeModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Tensorprodukt of M,N

Example:

```
LIB "modules.lib";
ring R = 0,(x,y,z),dp;
FreeModule F = freeModule(R,3,0);
F;
↦  3
↦ R
↦ free Module
↦ Degrees of the generators: {0} {0} {0}
↦
```

```
tensorFreeModule(F,F);
↦  9
↦ R
↦ free Module
↦
```

D.4.15.42 tensorProduct

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

D.4.15.43 pruneModule

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: M in a simplyfied presentation

Example:

```
LIB "modules.lib";
ring R = 0,(x,y,z),dp;
matrix a[2][3] = -x,-y^2,x^3,y,x,0;
matrix b[1][2] = x^2-y^3,xy;
Matrix A = a;
Matrix B = b;
Module M = coker(A);
Module N = coker(B);
Module H = hom(M,N);
H;
↦ subquotient (| 0  -y -y 0  -x  |, | -y3+x2 xy 0      0  |)
↦              | xy -x -x y3 -y2 |  | 0      0  -y3+x2 xy |
↦
↦
pruneModule(H);
↦ cokernel | 0 y2 x2 x  |
↦          | x 0  0  y3 |
↦
↦
```

D.4.15.44 hom

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: calculates Hom(M,N) as a subquotient and yields an interpretation for the elements

Example:

```
LIB "modules.lib";
"Example:";
↦ Example:
ring R = 0,(x,y,z),dp;
matrix a[2][3] = -x,-y^2,x^3,y,x,0;
matrix b[1][2] = x^2-y^3,xy;
Matrix A = a;
Matrix B = b;
Module M = coker(A);
Module N = coker(B);
hom(M,N);
```

```
↦ subquotient (| 0  -y -y 0  -x  |, | -y3+x2 xy 0      0  |)
↦              | xy -x -x y3 -y2 |  | 0      0  -y3+x2 xy |
↦
↦
```

D.4.15.45 kerHom

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: returns the kernel of the given homomorphism

Example:

```
LIB "modules.lib";
ring R=0,(x,y),(lp,c);
Matrix M=id(2);
Module src=image(M);
matrix rules[2][2]=x,y,xy,y2;
Module tar=coker(M);
Homomorphism f=homomorphism(rules,src,tar);
f;
↦ | x  y  |
↦ | xy y2 |
↦
↦ cokernel | 1 0 | <--- R^2
↦          | 0 1 |
↦
kerHom(f);
↦ image | 0  -1 |
↦       | -1 0  |
↦
↦
```

D.4.15.46 interpret

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: interpretation of V into some space that is stored in the interpretationlist of V.space

Example:

```
LIB "modules.lib";
"example 1:";
↦ example 1:
ring R = 0,(x,y),dp;
matrix a[1][2] = x,y;
Matrix A = a;
matrix b[1][2] = x2,y2;
Matrix B = b;
Module M = subquotient(A,B);
Module C = presentation(M);
Vector V = [x2,y4],C;
interpret(V);
↦ | y5+x3 |
↦
↦
"example 2:";
```

```
↦ example 2:
ring S;
matrix gens[2][3] = x2+y-3z4,y+xy,xyz+4,3+z2x,z3-3x+3,2+x+y+z7;
vector v = 2x*[gens[1..2,1]] + (y-z2)*[gens[1..2,2]] + 5*[gens[1..2,3]];
Matrix Gens = gens;
M = subquotient(Gens,zero(2,3));
M.interpretation = list(list(1,1,1),list());
V = v,M;
interpret(V);
↦ -z2+2x+y+5
```

D.4.15.47 interpretInv

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: interpretation of V into some Module N (inverse to interpret)

Example:

```
LIB "modules.lib";
ring R;
matrix gens[2][3] = x2,xy,4,z2x,3x+3,z;
vector v = 2x*[gens[1..2,1]] + (y-z2)*[gens[1..2,2]] + 5*[gens[1..2,3]];
Matrix Gens = gens;
Module S = subquotient(Gens,zero(2,3));
Module N = coker(id(3));
matrix E = N.generators.hom;
Vector E1 = [1,0,0],N;
Vector E2 = [0,1,0],N;
Vector E3 = [0,0,1],N;
S.interpretation = list(list(E1,E2,E3),list([gens[1..2,1]],[gens[1..2,2]],[gens[1..2
Vector V = v,S;
Vector W = interpret(V),N;
net(V);
↦ | -xyz2+2x3+xy2+20         |
↦ | 2x2z2-3xz2+3xy-3z2+3y+5z |
↦ 
Vector Vnew = interpretInv(W,S);
net(Vnew);
↦ | -xyz2+2x3+xy2+20         |
↦ | 2x2z2-3xz2+3xy-3z2+3y+5z |
↦ 
V==Vnew;
↦ 1
```

D.4.15.48 reduceIntChain

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Module C with minimized (or # steps) interpretation list

Example:

```
LIB "modules.lib";
ring R;
matrix a[5][4];
Module M = coker(a+1);
Module N = simplePrune(simplePrune(simplePrune(simplePrune(M))));
//reduceIntChain(N);
```

D.4.15.49 interpretElem

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: interpretation of a Vector with # steps or until can't interpret further

Example:

```
LIB "modules.lib";
ring R;
matrix a[5][4];
Module M = coker(a+1);
Module N = simplePrune(simplePrune(simplePrune(simplePrune(M))));
Vector V = [x+y],N;
interpretElem(V,3);
↦ | 0   |
↦ | 0   |
↦ | 0   |
↦ | x+y |
↦ 
↦ 
```

D.4.15.50 interpretList

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: interpretation of Elements in some abstract structure defined by the user or into a Module

Example:

```
LIB "modules.lib";
ring R;
matrix a[5][4];
Module M = coker(a+1);
Module N = simplePrune(simplePrune(simplePrune(simplePrune(M))));
Vector V = [x+y],N;
Vector W = [x2+y2+3*z2],N;
Vector U = [x+2y+27z],N;
list L = U,V,W;
//interpretList(L,3);
```

D.4.15.51 compareVectors

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Usage: compareVector(V1,V2); Vector V1,V2

Return: compares the given Vectors up tu equivalence

Example:

```
LIB "modules.lib";
ring r;
matrix m[2][1] = x,-y;
Module M = subquotient(id(2),m);
```

```
Vector V = [x,y],M;
Vector W = [0,2y],M;
Vector U = [x,y2],M;
compareVectors(V,W);
↦ 1
compareVectors(U,V);
↦ 0
```

D.4.15.52 simplePrune

Procedure from library `modules.lib` (see Section D.4.15 [modules_lib], page 1122).

Return: Simplyfied Module with reduced dimension

Example:

```
LIB "modules.lib";
ring R;
matrix a[5][4];
Module M = coker(a+1);
Module N = simplePrune(M);
```

D.4.16 modstd_lib

Library: modstd.lib

Purpose: Groebner bases of ideals/modules using modular methods

Authors: A. Hashemi Amir.Hashemi@lip6.fr
G. Pfister pfister@mathematik.uni-kl.de
H. Schoenemann hannes@mathematik.uni-kl.de
A. Steenpass steenpass@mathematik.uni-kl.de
S. Steidel steidel@mathematik.uni-kl.de

Overview: A library for computing Groebner bases of ideals/modules in the polynomial ring over the rational numbers using modular methods.

References:

E. A. Arnold: Modular algorithms for computing Groebner bases. J. Symb. Comp. 35, 403-419 (2003).

N. Idrees, G. Pfister, S. Steidel: Parallelization of Modular Algorithms. J. Symb. Comp. 46, 672-684 (2011).

Procedures:

D.4.16.1 modStd

Procedure from library `modstd.lib` (see Section D.4.16 [modstd_lib], page 1146).

Usage: modStd(I[, exactness]); I ideal/module, exactness int

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.
An optional parameter 'exactness' can be provided. If exactness = 1(default), the procedure computes a standard basis of I for sure; if exactness = 0, it computes a standard basis of I with high probability.

Example:

```
LIB "modstd.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
ideal J = modStd(I);
J;
↦ J[1]=x3+1/3x2+1/3
↦ J[2]=z4+1/5z2+4/5
↦ J[3]=y5+1/11y3+2/11
I = homog(I, t);
J = modStd(I);
J;
↦ J[1]=x3+1/3x2t+1/3t3
↦ J[2]=z4+1/5z2t2+4/5t4
↦ J[3]=y5+1/11y3t2+2/11t5
ring R2 = 0, (x,y,z), ds;
ideal I = jacob(x5+y6+z7+xyz);
ideal J = modStd(I, 0);
J;
↦ J[1]=xy+7z6
↦ J[2]=xz+6y5
↦ J[3]=yz+5x4
↦ J[4]=x5-7/5z7
↦ J[5]=y6-7/6z7
↦ J[6]=z8
ring R3 = 0, x(1..4), lp;
ideal I = cyclic(4);
ideal J1 = modStd(I, 1);   // default
ideal J2 = modStd(I, 0);
size(reduce(J1, J2));
↦ 0
size(reduce(J2, J1));
↦ 0
```

See also: Section D.2.6.1 [modular], page 883.

D.4.16.2 modGB

Procedure from library `modstd.lib` (see Section D.4.16 [modstd_lib], page 1146).

Usage: modGB(method, I[, exactness]); I ideal/module, exactness int method can be: std, slimgb, sba

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.
An optional parameter 'exactness' can be provided. If exactness = 1(default), the procedure computes a standard basis of I for sure; if exactness = 0, it computes a standard basis of I with high probability.

Example:

```
LIB "modstd.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
ideal J = modGB("slimgb",I);
J;
↦ J[1]=x3+1/3x2+1/3
↦ J[2]=z4+1/5z2+4/5
↦ J[3]=y5+1/11y3+2/11
I = homog(I, t);
J = modGB("slimgb",I);
J;
↦ J[1]=x3+1/3x2t+1/3t3
↦ J[2]=z4+1/5z2t2+4/5t4
↦ J[3]=y5+1/11y3t2+2/11t5
ring R3 = 0, x(1..4), lp;
ideal I = cyclic(4);
ideal J1 = modGB("slimgb",I, 1);   // default
ideal J2 = modGB("slimgb",I, 0);
size(reduce(J1, J2));
↦ 0
size(reduce(J2, J1));
↦ 0
```

See also: Section D.2.6.1 [modular], page 883.

D.4.16.3 modSyz

Procedure from library `modstd.lib` (see Section D.4.16 [modstd_lib], page 1146).

Usage: modSyz(I); I ideal/module

Return: a generating set of syzygies of I

Note: The procedure computes a the syzygy module of I (over the rational numbers) by using modular methods with high probability. The property of being a syzygy is tested.

Example:

```
LIB "modstd.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
modSyz(I);
↦ _[1]=z4*gen(1)-3/5x3*gen(3)-1/5x2*gen(3)+1/5z2*gen(1)-1/5*gen(3)+4/5*gen(\
   1)
↦ _[2]=y5*gen(1)-3/11x3*gen(2)+1/11y3*gen(1)-1/11x2*gen(2)-1/11*gen(2)+2/11\
   *gen(1)
↦ _[3]=y5*gen(3)-5/11z4*gen(2)+1/11y3*gen(3)-1/11z2*gen(2)+2/11*gen(3)-4/11\
   *gen(2)
simplify(syz(I),1);
↦ _[1]=z4*gen(1)-3/5x3*gen(3)-1/5x2*gen(3)+1/5z2*gen(1)-1/5*gen(3)+4/5*gen(\
   1)
↦ _[2]=y5*gen(1)-3/11x3*gen(2)+1/11y3*gen(1)-1/11x2*gen(2)-1/11*gen(2)+2/11\
   *gen(1)
↦ _[3]=y5*gen(3)-5/11z4*gen(2)+1/11y3*gen(3)-1/11z2*gen(2)+2/11*gen(3)-4/11\
   *gen(2)
```

See also: Section D.2.6.1 [modular], page 883.

D.4.16.4 modIntersect

Procedure from library `modstd.lib` (see Section D.4.16 [modstd_lib], page 1146).

Usage: modIntersect(I,J); I,J ideal/module

Return: a generating set of the intersection of I and J

Note: The procedure computes a the intersection of I and J
(over the rational numbers) by using modular methods
with high probability.
No additional tests are performed.

Example:

```
LIB "modstd.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
ideal J = maxideal(2);
modIntersect(I,J);
↦ _[1]=z4-12/5x3-4/5x2+1/5z2
↦ _[2]=x3t2+1/3x2t2+1/3t2
↦ _[3]=x3zt+1/3x2zt+1/3zt
↦ _[4]=x3yt+1/3x2yt+1/3yt
↦ _[5]=x4t+1/3x3t+1/3xt
↦ _[6]=x3z2+1/3x2z2+1/3z2
↦ _[7]=x3yz+1/3x2yz+1/3yz
↦ _[8]=x4z+1/3x3z+1/3xz
↦ _[9]=y5-6/11x3+1/11y3-2/11x2
↦ _[10]=x3y2+1/3x2y2+1/3y2
↦ _[11]=x4y+1/3x3y+1/3xy
↦ _[12]=x5+1/3x4+1/3x2
simplify(intersect(I,J),1);
↦ _[1]=z4-12/5x3-4/5x2+1/5z2
↦ _[2]=x3t2+1/3x2t2+1/3t2
↦ _[3]=x3zt+1/3x2zt+1/3zt
↦ _[4]=x3yt+1/3x2yt+1/3yt
↦ _[5]=x4t+1/3x3t+1/3xt
↦ _[6]=x3z2+1/3x2z2+1/3z2
↦ _[7]=x3yz+1/3x2yz+1/3yz
↦ _[8]=x4z+1/3x3z+1/3xz
↦ _[9]=y5-6/11x3+1/11y3-2/11x2
↦ _[10]=x3y2+1/3x2y2+1/3y2
↦ _[11]=x4y+1/3x3y+1/3xy
↦ _[12]=x5+1/3x4+1/3x2
```

See also: Section D.2.6.1 [modular], page 883.

D.4.17 monomialideal_lib

Library: monomialideal.lib

Purpose: Primary and irreducible decompositions of monomial ideals

Authors: I.Bermejo, ibermejo@ull.es
E.Garcia-Llorente, evgarcia@ull.es
Ph.Gimenez, pgimenez@agt.uva.es

Overview: A library for computing a primary and the irreducible decompositions of a monomial ideal using several methods.
In this library we also take advantage of the fact that the ideal is monomial to make some computations that are Grobner free in this case (radical, intersection, quotient...).

Procedures:

D.4.17.1 isMonomial

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isMonomial (I); I ideal.

Return: 1, if I is monomial ideal; 0, otherwise.

Assume: I is an ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y, w^2*x^2*y^2*z^2 - y^3*z+x^4*z^4*t^4, w*x*y*z*t - w*x^6*y^5*z^4, x
isMonomial(I);
↦ 1
ideal J = w^3*x*y + x^3*y^9*t^3, w^2*x^2*y^2*z^2 - y^3*z, w*x*y*z*t - w*x^6*y^5*z^4,
isMonomial(J);
↦ 0
```

D.4.17.2 minbaseMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: minbaseMon (I); I ideal.

Return: an ideal, the minimal monomial generators of I.
(-1 if the generators of I are not monomials)

Assume: I is an ideal generated by a list of monomials of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y, w^2*x^2*y^2*z^2, y^3*z,x^4*z^4*t^4, w*x*y*z*t,w*x^6*y^5*z^4, x^2*z
minbaseMon(I);
↦ _[1]=w3xy
↦ _[2]=y3z
↦ _[3]=wxyzt
↦ _[4]=x2z4t3
↦ _[5]=x2y2z2
```

D.4.17.3 gcdMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: gcdMon (f,g); f,g polynomials.

Return: a monomial, the greatest common divisor of f and g.

Assume: f and g are monomials of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(x,y,z,t),dp;
poly f = x^3*z^5*t^2;
poly g = y^6*z^3*t^3;
gcdMon(f,g);
↦ z3t2
```

D.4.17.4 lcmMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: lcmMon (f,g); f,g polynomials.

Return: a monomial,the least common multiple of f and g.

Assume: f,g are monomials of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(x,y,z,t),dp;
poly f = x^3*z^5*t^2;
poly g = y^6*z^3*t^3;
lcmMon(f,g);
↦ x3y6z5t3
```

D.4.17.5 membershipMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: membershipMon(f,I); f polynomial, I ideal.

Return: 1, if f lies in I; 0 otherwise.
(-1 if I and f are nonzero and I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I =  w*x, x^2, y*z*t, y^5*t;
poly f =  3*x^2*y + 6*t^5*z*y^6 - 4*x^2 + 8*w*x^5*y^6 - 10*y^10*t^10;
membershipMon(f,I);
↦ 1
poly g = 3*w^2*t^3 - 4*y^3*z*t^3 - 2*x^2*y^5*t + 4*x*y^3;
membershipMon(g,I);
↦ 0
```

D.4.17.6 intersectMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: intersectMon (I,J); I,J ideals.

Return: an ideal, the intersection of I and J.
(it returns -1 if I or J are not monomial ideals)

Assume: I,J are monomial ideals of the basering.

Note: the minimal monomial generating set is returned.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y,w*x*y*z*t,x^2*y^2*z^2,x^2*z^4*t^3,y^3*z;
ideal J = w*x, x^2, y*z*t, y^5*t;
intersectMon (I,J);
```

```
↦ _[1]=y3zt
↦ _[2]=wxyzt
↦ _[3]=w3xy
↦ _[4]=x2y2z2
↦ _[5]=x2z4t3
↦ _[6]=x2y3z
↦ _[7]=wxy3z
```

D.4.17.7 quotientMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: quotientMon (I,J); I,J ideals.

Return: an ideal, the quotient I:J.
(returns -1 if I or J is not monomial)

Assume: I,J are monomial ideals of the basering.

Note: the minimal monomial generating set is returned.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y,w*x*y*z*t,x^2*y^2*z^2,x^2*z^4*t^3,y^3*z;
ideal J = w*x, x^2, y*z*t, y^5*t;
quotientMon (I,J);
↦ _[1]=y2z2t
↦ _[2]=y3z
↦ _[3]=xy2z2
↦ _[4]=x2z4t3
↦ _[5]=wy2zt
↦ _[6]=wxz4t3
↦ _[7]=wxyzt
↦ _[8]=w2y2z2
↦ _[9]=w3y2z
↦ _[10]=w3xy
```

D.4.17.8 radicalMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: radicalMon(I); I ideal

Return: an ideal, the radical ideal of the ideal I.
(returns -1 if I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Note: the minimal monomial generating set is returned.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y,w*x*y*z*t,x^2*y^2*z^2,x^2*z^4*t^3,y^3*z;
radicalMon(I);
↦ _[1]=yz
↦ _[2]=xzt
↦ _[3]=wxy
```

D.4.17.9 isprimeMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isprimeMon (I); I ideal

Return: 1, if I is prime; 0, otherwise.
(returns -1 if I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w,y,t;
isprimeMon (I);
↦ 1
ideal J = w,y,t,x*z;
isprimeMon (J);
↦ 0
```

D.4.17.10 isprimaryMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isprimaryMon (I); I ideal

Return: 1, if I is primary; 0, otherwise.
(returns -1 if I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^4,x^3,z^2,t^5,x*t,w*x^2*z;
isprimaryMon (I);
↦ 1
ideal J = w^4,x^3,z^2,t^5,x*t,w*x^2*z,y^3*t^3;
isprimaryMon (J);
↦ 0
```

D.4.17.11 isirreducibleMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isirreducibleMon(I); I ideal

Return: 1, if I is irreducible; 0, otherwise.
(return -1 if I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^4,x^3,z^2,t^5;
isirreducibleMon (I);
```

```
↦ 1
ideal J = w^4*x,x^3,z^2,t^5;
isirreducibleMon (J);
↦ 0
```

D.4.17.12 isartinianMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isartinianMon(I); I ideal.

Return: 1, if ideal is artinian; 0, otherwise.
(return -1 if ideal I is not a monmomial ideal).

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^4,x^3,y^4,z^2,t^6,w^2*x^2*y,w*z*t^4,x^2*y^3,z^2*t^5;
isartinianMon (I);
↦ 1
ideal J = w^4,x^3,y^4,z^2,w^2*x^2*y,w*z*t^4,x^2*y^3,z^2*t^5;
isartinianMon (J);
↦ 0
```

D.4.17.13 isgenericMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: isgenericMon(I); I ideal.

Return: 1, if ideal is generic; 0, otherwise.
(return -1 if ideal I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^4,x^3,y^4,z^2,w^2*x^2*y,w*z*t^4,x*y^3,z*t^5;
isgenericMon (I);
↦ 0
ideal J = w^4,x^3,y^4,z^3,w^2*x^2*y,w*z*t^4,x*y^3,z^2*t^5;
isgenericMon (J);
↦ 1
```

D.4.17.14 dimMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: dimMon (I); I ideal

Return: an integer, the dimension of the affine variety defined by the ideal I.
(returns -1 if I is not a monomial ideal)

Assume: I is a monomial ideal of the basering.

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z,t),lp;
ideal I = w^3*x*y,w*x*y*z*t,x^2*y^2*z^2,x^2*z^4*t^3,y^3*z;
dimMon (I);
↦ 3
ideal J = w^4,x^3,y^4,z^2,t^6,w^2*x^2*y,w*z*t^4,x^2*y^3,z*t^5;
dimMon (J);
↦ 0
```

D.4.17.15 irreddecMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: irreddecMon (I[,alg]); I ideal, alg string.

Return: list, the irreducible components of the monomial ideal I. (returns -1 if I is not a monomial ideal).

Assume: I is a monomial ideal of the basering k[x(1)..x(n)].

Note: This procedure returns the irreducible decomposition of I. One may call the procedure with different algorithms using the optional argument 'alg':
- the direct method following Vasconcelos' book (alg=vas) - via the Alexander dual and using doble dual (alg=add), - via the Alexander dual and quotients following E. Miller (alg=ad),
- the formula of irreducible components (alg=for),
- via the Scarf complex following Milowski (alg=mil),
- using the label algorithm of Roune (alg=lr),
- using the algorithm of Gao-Zhu (alg=gz).
- using the slice algorithm of Roune (alg=sr).

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z),Dp;
ideal I = w^3*x*y,w*x*y*z,x^2*y^2*z^2,x^2*z^4,y^3*z;
// Vasconcelos
irreddecMon (I,"vas");
↦ [1]:
↦    _[1]=y
↦    _[2]=x2
↦ [2]:
↦    _[1]=w
↦    _[2]=z2
↦    _[3]=y3
↦ [3]:
↦    _[1]=y
↦    _[2]=z4
↦ [4]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [5]:
↦    _[1]=w
↦    _[2]=y2
↦    _[3]=z4
↦ [6]:
↦    _[1]=z
↦    _[2]=w3
↦ [7]:
↦    _[1]=z
↦    _[2]=x
↦ [8]:
↦    _[1]=x
↦    _[2]=y3
// Alexander Dual
irreddecMon (I,"ad");
↦ [1]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z2
↦ [2]:
↦    _[1]=w
↦    _[2]=y2
↦    _[3]=z4
↦ [3]:
↦    _[1]=x
↦    _[2]=z
↦ [4]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [5]:
↦    _[1]=w3
↦    _[2]=z
↦ [6]:
↦    _[1]=x2
↦    _[2]=y
↦ [7]:
↦    _[1]=y
↦    _[2]=z4
↦ [8]:
↦    _[1]=x
↦    _[2]=y3
// Scarf Complex
irreddecMon (I,"mil");
↦ [1]:
↦    _[1]=y
↦    _[2]=z4
↦ [2]:
↦    _[1]=w3
↦    _[2]=z
↦ [3]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z2
↦ [4]:
↦    _[1]=w
↦    _[2]=y2
↦    _[3]=z4
↦ [5]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [6]:
↦    _[1]=x
↦    _[2]=y3
↦ [7]:
↦    _[1]=x2
↦    _[2]=y
↦ [8]:
↦    _[1]=x
↦    _[2]=z
// slice algorithm
irreddecMon(I,"sr");
↦ [1]:
↦    _[1]=y
↦    _[2]=z4
↦ [2]:
↦    _[1]=x2
↦    _[2]=y
↦ [3]:
↦    _[1]=x
↦    _[2]=z
↦ [4]:
↦    _[1]=x
↦    _[2]=y3
↦ [5]:
↦    _[1]=w3
↦    _[2]=z
↦ [6]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z2
↦ [7]:
↦    _[1]=w
↦    _[2]=y2
↦    _[3]=z4
↦ [8]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
```

D.4.17.16 primdecMon

Procedure from library `monomialideal.lib` (see Section D.4.17 [monomialideal_lib], page 1149).

Usage: primdecMon (I[,alg]); I ideal, alg string

Return: list, the components in a minimal primary decomposition of I. (returns -1 if I is not a monomial ideal).

Assume: I is a monomial ideal of the basering k[x(1)..x(n)].

Note: This procedure returns a minimal primary decomposition of I. One may call the procedure with different algorithms using the optional argument 'alg':
- the direct method for a primary decomposition following Vasconcelos' book (alg=vp),
- from the irreducible decomposition obtained via the direct method following Vasconcelos' book (alg=vi),
- from the irreducible decomposition obtained via the
Alexander dual and using doble dual (alg=add),
- from the irreducible decomposition obtained via the
Alexander dual and quotients following E. Miller (alg=ad), - from the irreducible decomposition obtained
via .... (alg=for),
- from the irreducible decomposition obtained via the Scarf complex following Milowski (alg=mil),
- from the irreducible decomposition obtained using the label algorithm of Roune (alg=lr),
- from the irreducible decomposition obtained using the algorithm of Gao-Zhu (alg=gz),
- from the irreducible decomposition obtained using the slice algorithm of Roune (alg=sr).

Example:

```
LIB "monomialideal.lib";
ring R = 0,(w,x,y,z),Dp;
ideal I = w^3*x*y,w*x*y*z,x^2*y^2*z^2,x^2*z^4,y^3*z;
// Vasconcelos para primaria
primdecMon(I,"vp");
↦ [1]:
↦    _[1]=x2
↦    _[2]=y3
↦    _[3]=wxy
↦    _[4]=w3
↦ [2]:
↦    _[1]=xy
↦    _[2]=x2
↦    _[3]=y3
↦ [3]:
↦    _[1]=z
↦    _[2]=x
↦ [4]:
↦    _[1]=y3
↦    _[2]=wyz
↦    _[3]=w3
↦    _[4]=z4
↦    _[5]=y2z2
↦ [5]:
↦    _[1]=y
↦    _[2]=z4
↦ [6]:
↦    _[1]=z
↦    _[2]=w3
// Alexander dual
```

```
primdecMon(I,"add");
↦ [1]:
↦    _[1]=y
↦    _[2]=z4
↦ [2]:
↦    _[1]=xy
↦    _[2]=x2
↦    _[3]=y3
↦ [3]:
↦    _[1]=w3
↦    _[2]=z
↦ [4]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [5]:
↦    _[1]=x
↦    _[2]=z
↦ [6]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z4
↦    _[4]=y2z2
// label algorithm
primdecMon(I,"lr");
↦ [1]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [2]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z4
↦    _[4]=y2z2
↦ [3]:
↦    _[1]=w3
↦    _[2]=z
↦ [4]:
↦    _[1]=x
↦    _[2]=z
↦ [5]:
↦    _[1]=xy
↦    _[2]=x2
↦    _[3]=y3
↦ [6]:
↦    _[1]=y
↦    _[2]=z4
//slice algorithm
primdecMon(I,"sr");
↦ [1]:
↦    _[1]=w
↦    _[2]=x2
↦    _[3]=y3
↦ [2]:
↦    _[1]=w
↦    _[2]=y3
↦    _[3]=z4
↦    _[4]=y2z2
↦ [3]:
↦    _[1]=w3
↦    _[2]=z
↦ [4]:
↦    _[1]=x
↦    _[2]=z
↦ [5]:
↦    _[1]=xy
↦    _[2]=x2
↦    _[3]=y3
↦ [6]:
↦    _[1]=y
↦    _[2]=z4
```

D.4.18 mprimdec_lib

Library: mprimdec.lib

Purpose: procedures for primary decomposition of modules

Authors: Alexander Dreyer, dreyer@mathematik.uni-kl.de; adreyer@web.de

Overview: Algorithms for primary decomposition for modules based on the algorithms of Gianni, Trager and Zacharias and
Shimoyama and Yokoyama (generalization of the latter suggested by Hans-Gert Graebe, Leipzig)
using elements of primdec.lib

Remark: These procedures are implemented to be used in characteristic 0.
They also work in positive characteristic >> 0.
In small characteristic and for algebraic extensions, the procedures via Gianni, Trager, Zacharias may not terminate.

Procedures:

D.4.18.1 separator

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: separator(l); list l of prime ideals

Return: list sepList;
a list of separators of the prime ideals in l,
i.e. polynomials p_ij, s.th. p_ij is in l[j],
for all l[j] not contained in l[i]
but p_ij is not in l[i]

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
ideal i=(x2y,xz2,y2z,z3);
```

```
list l=minAssGTZ(i);
list sepL=separator(l);
sepL;
↦ [1]:
↦    x
↦ [2]:
↦    y
```

D.4.18.2 PrimdecA

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: PrimdecA (N[, i]); module N, int i

Return: list l
a (not necessarily minimal) primary decomposition of N computed by a generalized version of the algorithm of Shimoyama/Yokoyama,
if i!=0 is given, the factorizing Groebner is used to compute the isolated primes

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=x*gen(1)+ y*gen(2),
x*gen(1)-x2*gen(2);
list l=PrimdecA(N);
l;
↦ [1]:
↦    [1]:
↦       _[1]=x*gen(1)+y*gen(2)
↦       _[2]=x*gen(2)-gen(1)
↦    [2]:
↦       _[1]=x2+y
↦ [2]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=x*gen(1)
↦    [2]:
↦       _[1]=x
↦ [3]:
↦    [1]:
↦       _[1]=y*gen(1)
↦       _[2]=y*gen(2)
↦       _[3]=x*gen(1)
↦       _[4]=x*gen(2)
↦    [2]:
↦       _[1]=y
↦       _[2]=x
```

D.4.18.3 PrimdecB

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: PrimdecB (N, p); pseudo-primary module N, isolated prime ideal p

Return: list l
a (not necessarily minimal) primary decomposition of N

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=y*gen(1),y2*gen(2),yz*gen(2),yx*gen(2);
ideal p=y;
list l=PrimdecB(N,p);
l;
↦ [1]:
↦    [1]:
↦       _[1]=y*gen(1)
↦       _[2]=y*gen(2)
↦    [2]:
↦       _[1]=y
↦ [2]:
↦    [1]:
↦       _[1]=y*gen(1)
↦       _[2]=y*gen(2)
↦       _[3]=x*gen(1)
↦       _[4]=x*gen(2)
↦    [2]:
↦       _[1]=y
↦       _[2]=x
↦ [3]:
↦    [1]:
↦       _[1]=z*gen(1)
↦       _[2]=z*gen(2)
↦       _[3]=y*gen(1)
↦       _[4]=x*gen(1)
↦       _[5]=x*gen(2)
↦       _[6]=y2*gen(2)
↦    [2]:
↦       _[1]=z
↦       _[2]=y
↦       _[3]=x
```

D.4.18.4 modDec

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: modDec (N[, i]); module N, int i

Return: list l
a minimal primary decomposition of N
computed by an generalized version of the algorithm of Shimoyama/Yokoyama,
if i=1 is given, the factorizing Groebner basis algorithm is used internally.

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=x*gen(1)+ y*gen(2),
x*gen(1)-x2*gen(2);
list l=modDec(N);
l;
↦ [1]:
```

```
↦      [1]:
↦         _[1]=x*gen(1)+y*gen(2)
↦         _[2]=x*gen(2)-gen(1)
↦      [2]:
↦         _[1]=x2+y
↦   [2]:
↦      [1]:
↦         _[1]=gen(2)
↦         _[2]=x*gen(1)
↦      [2]:
↦         _[1]=x
```

D.4.18.5 zeroMod

Procedure from library mprimdec.lib (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: zeroMod (N[, check]); zero-dimensional module N[, module check]

Return: list l
the minimal primary decomposition of a zero-dimensional module N, computed by a generalized version of the algorithm of Gianni, Trager and Zacharias

Note: if the parameter check is given, only components not containing check are computed

Example:

```
LIB "mprimdec.lib";
ring r=0,z,dp;
module N=z*gen(1),(z-1)*gen(2),(z+1)*gen(3);
list l=zeroMod(N);
↦ 2
l;
↦   [1]:
↦      [1]:
↦         _[1]=gen(2)
↦         _[2]=gen(3)
↦         _[3]=z*gen(1)
↦      [2]:
↦         _[1]=z
↦   [2]:
↦      [1]:
↦         _[1]=gen(1)
↦         _[2]=gen(3)
↦         _[3]=z*gen(2)-gen(2)
↦      [2]:
↦         _[1]=z-1
↦   [3]:
↦      [1]:
↦         _[1]=gen(1)
↦         _[2]=gen(2)
↦         _[3]=z*gen(3)+gen(3)
↦      [2]:
↦         _[1]=z+1
```

D.4.18.6 GTZmod

Procedure from library mprimdec.lib (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: GTZmod (N[, check]); module N[, module check]

Return: list l
the minimal primary decomposition of the module N,
computed by a generalized version of the algorithm of Gianni, Trager and Zacharias

Note: if the parameter check is given, only components not containing check are computed

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=x*gen(1)+ y*gen(2),
x*gen(1)-x2*gen(2);
list l=GTZmod(N);
↦ 2
l;
↦ [1]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=x*gen(1)
↦    [2]:
↦       _[1]=x
↦ [2]:
↦    [1]:
↦       _[1]=x*gen(1)+y*gen(2)
↦       _[2]=x*gen(2)-gen(1)
↦    [2]:
↦       _[1]=x2+y
```

D.4.18.7 dec1var

Procedure from library mprimdec.lib (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: dec1var (N); zero-dimensional module N[, module check]

Return: list l
the minimal primary decomposition of a submodule N of R^s if nvars(R)=1

Note: if the parameter check is given, only components not containing check are computed

Example:

```
LIB "mprimdec.lib";
ring r=0,z,dp;
module N=z*gen(1),(z-1)*gen(2),(z+1)*gen(3);
list l=dec1var(N);
l;
↦ [1]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=gen(3)
↦       _[3]=z*gen(1)
↦    [2]:
↦       _[1]=z
↦ [2]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(3)
↦       _[3]=z*gen(2)-gen(2)
↦    [2]:
↦       _[1]=z-1
↦ [3]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(2)
↦       _[3]=z*gen(3)+gen(3)
↦    [2]:
↦       _[1]=z+1
```

D.4.18.8 annil

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: annil(N); modul N

Return: ideal ann=std(quotient(N,freemodule(nrows(N))));
the annihilator of M/N in the basering

Note: ann is a std basis in the basering

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=x*gen(1), y*gen(2);
ideal ann=annil(N);
ann;
↦ ann[1]=xy
```

D.4.18.9 splitting

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: splitting(N[,check[, ann]]); modul N, module check, ideal ann

Return: (l, check) list l, module check
the elements of l consists of quadruples, where
[1] is of type module, [2], [3] and [4] are of type ideal, s.th. the intersection of the modules is equal to the zero-dimensional module N, furthermore l[j][3]=annil(l[j][1]) and l[j][4] contains internal ideal data;
if l[j][2]!=0 then the module l[j][1] is primary
with associated prime l[j][2], and check=intersect(check, l[j][1]) is computed

Note: if the parameter check is given, only components not containing check are computed;
if ann is given, ann is used instead of annil(N)

Example:

```
LIB "mprimdec.lib";
ring r=0,z,lp;
module N=z*gen(1), (z+1)*gen(2);
N=std(N);
list l; module check;
(l, check)=splitting(N);
l;
```

```
↦ [1]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=z*gen(1)
↦    [2]:
↦       _[1]=z
↦    [3]:
↦       _[1]=z
↦    [4]:
↦       _[1]=z
↦ [2]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=z*gen(2)+gen(2)
↦    [2]:
↦       _[1]=z+1
↦    [3]:
↦       _[1]=z+1
↦    [4]:
↦       _[1]=z+1
check;
↦ check[1]=z*gen(2)+gen(2)
↦ check[2]=z*gen(1)
```

D.4.18.10 primTest

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: primTest(i[, p]); a zero-dimensional ideal i, irreducible poly p in i

Return: if i is neither prime nor homogeneous then ideal(0) is returned, otherwise radical(i)

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),lp;
ideal i=x+1,y-1,z;
i=std(i);
ideal primId=primTest(i,z);
primId;
↦ primId[1]=z
↦ primId[2]=y-1
↦ primId[3]=x+1
i=x,z2,yz,y2;
i=std(i);
primId=primTest(i);
primId;
↦ primId[1]=x
↦ primId[2]=y
↦ primId[3]=z
```

D.4.18.11 preComp

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: preComp(N,check[, ann]); modul N, module check, ideal ann

Return: (l, check) list l, module check
the elements of l consists of a triple with
[1] of type module [2] and [3] of type ideal
s.th. the intersection of the modules is equal to the zero-dimensional module N, furthermore l[j][3]=annil(l[j][1]) if l[j][2]!=0 then the module l[j][1] is primary
with associated prime l[j][2],
and check=intersect(check, l[j][1]) is computed

Note: only components not containing check are computed;
if ann is given, ann is used instead of annil(N)

Example:

```
LIB "mprimdec.lib";
ring r=0,z,lp;
module N=z*gen(1), (z+1)*gen(2);
N=std(N);
list l; module check;
(l, check)=preComp(N,freemodule(2));
l;
↦ [1]:
↦    [1]:
↦       _[1]=z*gen(1)
↦       _[2]=gen(2)
↦    [2]:
↦       _[1]=z
↦    [3]:
↦       _[1]=z
↦ [2]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=z*gen(2)+gen(2)
↦    [2]:
↦       _[1]=z+1
↦    [3]:
↦       _[1]=z+1
check;
↦ check[1]=z*gen(1)
↦ check[2]=z*gen(2)+gen(2)
```

D.4.18.12 indSet

Procedure from library mprimdec.lib (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: indSet(i); i ideal

Return: list with two entries
both are lists of new varstrings with the dependent variables, the independent set, the ordstring with the corresp. block ordering, and the integer where the independent set starts in the varstring

Note: the first entry gives the strings for all maximal independent sets the second gives the strings for the independent sets,
which cannot be enhanced

Example:

```
LIB "mprimdec.lib";
ring s1=(0,x,y),(a,b,c,d,e,f,g),lp;
ideal i=ea-fbg,fa+be,ec-fdg,fc+de;
i=std(i);
list  l=indSet(i);
l;
↦ [1]:
↦    [1]:
↦       [1]:
↦          e,f
↦       [2]:
↦          a,b,c,d,g
↦       [3]:
↦          (C,dp(2),dp)
↦       [4]:
↦          5
↦ [2]:
↦    [1]:
↦       [1]:
↦          a,b,c,d
↦       [2]:
↦          e,f,g
↦       [3]:
↦          (C,dp(4),dp)
↦       [4]:
↦          3
↦    [2]:
↦       [1]:
↦          a,c,e
↦       [2]:
↦          b,d,f,g
↦       [3]:
↦          (C,dp(3),dp)
↦       [4]:
↦          4
```

D.4.18.13 GTZopt

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: GTZopt (N[, check]); module N[, module check]

Return: list l
the minimal primary decomposition of the module N,
computed by a generalized and optimized version of
the algorithm of Gianni, Trager and Zacharias

Note: if the parameter check is given, only components
not containing check are computed

Example:

```
LIB "mprimdec.lib";
ring r=0,(x,y,z),dp;
module N=x*gen(1)+ y*gen(2),
x*gen(1)-x2*gen(2);
```

```
list l=GTZopt(N);
l;
↦ [1]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=x*gen(1)
↦    [2]:
↦       _[1]=x
↦ [2]:
↦    [1]:
↦       _[1]=x*gen(1)+y*gen(2)
↦       _[2]=x*gen(2)-gen(1)
↦    [2]:
↦       _[1]=x2+y
```

D.4.18.14 zeroOpt

Procedure from library `mprimdec.lib` (see Section D.4.18 [mprimdec_lib], page 1160).

Usage: zeroOpt (N[, check]); zero-dimensional module N[, module check]

Return: list l
the minimal primary decomposition of a zero-dimensional module N, computed by a generalized and optimized version of the algorithm of Gianni, Trager and Zacharias

Note: if the parameter check is given, only components
not containing check are computed

Example:

```
LIB "mprimdec.lib";
ring r=0,z,dp;
module N=z*gen(1),(z-1)*gen(2),(z+1)*gen(3);
list l=zeroOpt(N);
l;
↦ [1]:
↦    [1]:
↦       _[1]=z*gen(1)
↦       _[2]=gen(2)
↦       _[3]=gen(3)
↦    [2]:
↦       _[1]=z
↦ [2]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=z*gen(2)-gen(2)
↦       _[3]=gen(3)
↦    [2]:
↦       _[1]=z-1
↦ [3]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(2)
↦       _[3]=z*gen(3)+gen(3)
↦    [2]:
↦       _[1]=z+1
```

D.4.19 mregular_lib

Library: mregular.lib

Purpose: Castelnuovo-Mumford regularity of homogeneous ideals

Authors: I.Bermejo, ibermejo@ull.es
Ph.Gimenez, pgimenez@agt.uva.es
G.-M.Greuel, greuel@mathematik.uni-kl.de

Overview: A library for computing the Castelnuovo-Mumford regularity of a homogeneous ideal that DOES NOT require the computation of a minimal graded free resolution of the ideal.
It also determines depth(basering/ideal) and satiety(ideal). The procedures are based on 3 papers by Isabel Bermejo and Philippe Gimenez: 'On Castelnuovo-Mumford regularity of projective curves' Proc.Amer.Math.Soc. 128(5) (2000), 'Computing the Castelnuovo-Mumford regularity of some subschemes of Pn using quotients of monomial ideals', Proceedings of MEGA-2000, J. Pure Appl. Algebra 164 (2001), and 'Saturation and Castelnuovo-Mumford regularity', Preprint (2004).

Procedures:

D.4.19.1 regIdeal

Procedure from library `mregular.lib` (see Section D.4.19 [mregular_lib], page 1170).

Usage: regIdeal (i[,e]); i ideal, e integer

Return: an integer, the Castelnuovo-Mumford regularity of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal of the basering S=K[x(0)..x(n)]. e=0: (default)
If K is an infinite field, makes random changes of coordinates. If K is a finite field, works over a transcendental extension. e=1: Makes random changes of coordinates even when K is finite. It works if it terminates, but may result in an infinite loop. After 30 loops, a warning message is displayed and -1 is returned.

Note: If printlevel > 0 (default = 0), additional info is displayed: dim(S/i), depth(S/i) and end(H^(depth(S/i))(S/i)) are computed, and an upper bound for the a-invariant of S/i is given. The algorithm also determines whether the regularity is attained or not at the last step of a minimal graded free resolution of i, and if the answer is positive, the regularity of the Hilbert function of S/i is given.

Example:

```
LIB "mregular.lib";
ring r=0,(x,y,z,t,w),dp;
ideal i=y2t,x2y-x2z+yt2,x2y2,xyztw,x3z2,y5+xz3w-x2zw2,x7-yt2w4;
regIdeal(i);
↦ 10
regIdeal(lead(std(i)));
↦ 13
// Additional information is displayed if you change printlevel (=1);
```

D.4.19.2 depthIdeal

Procedure from library **mregular.lib** (see Section D.4.19 [mregular_lib], page 1170).

Usage: depthIdeal (i[,e]); i ideal, e integer

Return: an integer, the depth of S/i where S=K[x(0)..x(n)] is the basering. (returns -1 if i is not homogeneous or if i=(1))

Assume: i is a proper homogeneous ideal.
e=0: (default)
If K is an infinite field, makes random changes of coordinates. If K is a finite field, works over a transcendental extension. e=1: Makes random changes of coordinates even when K is finite. It works if it terminates, but may result in an infinite loop. After 30 loops, a warning message is displayed and -1 is returned.

Note: If printlevel > 0 (default = 0), dim(S/i) is also displayed.

Example:

```
LIB "mregular.lib";
ring r=0,(x,y,z,t,w),dp;
ideal i=y2t,x2y-x2z+yt2,x2y2,xyztw,x3z2,y5+xz3w-x2zw2,x7-yt2w4;
depthIdeal(i);
↦ 1
depthIdeal(lead(std(i)));
↦ 0
// Additional information is displayed if you change printlevel (=1);
```

D.4.19.3 satiety

Procedure from library **mregular.lib** (see Section D.4.19 [mregular_lib], page 1170).

Usage: satiety (i[,e]); i ideal, e integer

Return: an integer, the satiety of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal of the basering S=K[x(0)..x(n)]. e=0: (default)
The satiety is computed determining the fresh elements in the socle of i. It works over arbitrary fields.
e=1: Makes random changes of coordinates to find a monomial ideal with same satiety. It works over infinite fields only. If K is finite, it works if it terminates, but may result in an infinite loop. After 30 loops, a warning message is displayed and -1 is returned.

Theory: The satiety, or saturation index, of a homogeneous ideal i is the least integer s such that, for all d>=s, the degree d part of the ideals i and isat=sat(i,maxideal(1)) coincide.

Note: If printlevel > 0 (default = 0), dim(S/i) is also displayed.

Example:

```
LIB "mregular.lib";
ring r=0,(x,y,z,t,w),dp;
ideal i=y2t,x2y-x2z+yt2,x2y2,xyztw,x3z2,y5+xz3w-x2zw2,x7-yt2w4;
satiety(i);
↦ 0
ideal I=lead(std(i));
satiety(I);   // First  method: direct computation
```

```
↦ 12
satiety(I,1); // Second method: doing changes of coordinates
↦ 12
// Additional information is displayed if you change printlevel (=1);
```

D.4.19.4 regMonCurve

Procedure from library `mregular.lib` (see Section D.4.19 [mregular_lib], page 1170).

Usage: regMonCurve (a0,...,an) ; ai integers with a0=0 < a1 < ... < an=:d

Return: an integer, the Castelnuovo-Mumford regularity of the projective monomial curve C in Pn(K) parametrically defined by
x(0) = t^d , x(1) = s^(a1)t^(d-a1) , ... , x(n) = s^d where K is the field of complex numbers.
(returns -1 if a0=0 < a1 < ... < an is not satisfied)

Assume: a0=0 < a1 < ... < an are integers.

Notes: 1. The defining ideal of the curve C, I in S=K[x(0),...,x(n)], is determined by elimination.
2. The procedure regIdeal has been improved in this case since one knows beforehand that the monomial ideal J=lead(std(I)) is of nested type if the monomial ordering is dp, and that
reg(C)=reg(J) (see preprint 'Saturation and Castelnuovo-Mumford regularity' by Bermejo-Gimenez, 2004).
3. If printlevel > 0 (default = 0) additional info is displayed: - It says whether C is arithmetically Cohen-Macaulay or not. - If C is not arith. Cohen-Macaulay, end(H^1(S/I)) is computed and an upper bound for the a-invariant of S/I is given. - It also determines one step of the minimal graded free resolution (m.g.f.r.) of I where the regularity is attained and gives the value of the regularity of the Hilbert function of S/I when reg(I) is attained at the last step of a m.g.f.r.

Example:

```
LIB "mregular.lib";
// The 1st example is the twisted cubic:
regMonCurve(0,1,2,3);
↦ 2
// The 2nd. example is the non arithm. Cohen-Macaulay monomial curve in P4
// parametrized by: x(0)-s6,x(1)-s5t,x(2)-s3t3,x(3)-st5,x(4)-t6:
regMonCurve(0,1,3,5,6);
↦ 3
// Additional information is displayed if you change printlevel (=1);
```

D.4.19.5 NoetherPosition

Procedure from library `mregular.lib` (see Section D.4.19 [mregular_lib], page 1170).

Usage: NoetherPosition (i); i ideal

Return: ideal such that, for the homogeneous linear transformation map phi=S,NoetherPosition(i);
one has that K[x(n-d+1),...,x(n)] is a Noether normalization of S/phi(i) where S=K[x(0),...x(n)] is the basering and d=dim(S/i). (returns -1 if i = (0) or (1)).

Assume: The field K is infinite and i is a nonzero proper ideal.

Note: 1. It works also if K is a finite field if it terminates, but may result in an infinite loop. If the procedure enters more than 30 loops, -1 is returned and a warning message is displayed.
2. If printlevel > 0 (default = 0), additional info is displayed: dim(S/i) and K[x(n-d+1),...,x(n)] are given.

Example:

```
LIB "mregular.lib";
ring r=0,(x,y,z,t,u),dp;
ideal i1=y,z,t,u; ideal i2=x,z,t,u; ideal i3=x,y,t,u; ideal i4=x,y,z,u;
ideal i5=x,y,z,t; ideal i=intersect(i1,i2,i3,i4,i5);
map phi=r,NoetherPosition(i);
phi;
↦ phi[1]=x
↦ phi[2]=y
↦ phi[3]=z
↦ phi[4]=t
↦ phi[5]=53x+27y-75z+45t+u
ring r5=5,(x,y,z,t,u),dp;
ideal i=imap(r,i);
map phi=r5,NoetherPosition(i);
phi;
↦ phi[1]=x
↦ phi[2]=y
↦ phi[3]=z
↦ phi[4]=t
↦ phi[5]=x-y+z-t+u
// Additional information is displayed if you change printlevel (=1);
```

D.4.19.6 is_NP

Procedure from library `mregular.lib` (see Section D.4.19 [mregular_lib], page 1170).

Usage: is_NP (i); i ideal

Return: 1 if K[x(n-d+1),...,x(n)] is a Noether normalization of S/i where S=K[x(0),...x(n)] is the basering, and d=dim(S/i), 0 otherwise.
(returns -1 if i=(0) or i=(1)).

Assume: i is a nonzero proper homogeneous ideal.

Note: 1. If i is not homogeneous and is_NP(i)=1 then K[x(n-d+1),...,x(n)] is a Noether normalization of S/i. The converse may be wrong if the ideal is not homogeneous.
2. is_NP is used in the procedures regIdeal, depthIdeal, satiety, and NoetherPosition.

Example:

```
LIB "mregular.lib";
ring r=0,(x,y,z,t,u),dp;
ideal i1=y,z,t,u; ideal i2=x,z,t,u; ideal i3=x,y,t,u; ideal i4=x,y,z,u;
ideal i5=x,y,z,t; ideal i=intersect(i1,i2,i3,i4,i5);
is_NP(i);
↦ 0
ideal ch=x,y,z,t,x+y+z+t+u;
map phi=ch;
is_NP(phi(i));
↦ 1
```

D.4.19.7 is_nested

Procedure from library `mregular.lib` (see Section D.4.19 [mregular_lib], page 1170).

Usage: is_nested (i); i monomial ideal

Return: 1 if i is of nested type, 0 otherwise.
(returns -1 if i=(0) or i=(1)).

Assume: i is a nonzero proper monomial ideal.

Notes: 1. The ideal must be monomial, otherwise the result has no meaning (so check this before using this procedure).
2. is_nested is used in procedures depthIdeal, regIdeal and satiety.
3. When i is a monomial ideal of nested type of S=K[x(0)..x(n)], the a-invariant of S/i coincides with the upper bound obtained using the procedure regIdeal with printlevel > 0.

Theory: A monomial ideal is of nested type if its associated primes are all of the form (x(0),...,x(i)) for some i<=n.
(see definition and effective criterion to check this property in the preprint 'Saturation and Castelnuovo-Mumford regularity' by Bermejo-Gimenez, 2004).

Example:

```
LIB "mregular.lib";
ring s=0,(x,y,z,t),dp;
ideal i1=x2,y3; ideal i2=x3,y2,z2; ideal i3=x3,y2,t2;
ideal i=intersect(i1,i2,i3);
is_nested(i);
↦ 0
ideal ch=x,y,z,z+t;
map phi=ch;
ideal I=lead(std(phi(i)));
is_nested(I);
↦ 1
```

D.4.20 nfmodstd_lib

Library: nfmodstd.lib

Purpose: Groebner bases of ideals in polynomial rings over algebraic number fields

Authors: D.K. Boku boku@mathematik.uni-kl.de
W. Decker decker@mathematik.uni-kl.de
C. Fieker fieker@mathematik.uni-kl.de

Overview: A library for computing the Groebner basis of an ideal in the polynomial ring over an algebraic number field Q(t) using the modular methods, where t is algebraic over the field of rational numbers Q. For the case Q(t) = Q, the procedure is inspired by Arnold [1]. This idea is then extended to the case t not in Q using factorization as follows:
Let f be the minimal polynomial of t.
For I, I' ideals in Q(t)[X], Q[X,t]/<f> respectively, we map I to I' via the map sending t to t + <f>. We first choose a prime p such that f has at least two factors in characteristic p and add each factor f_i to I' to obtain the ideal J'_i = I' + <f_i>. We then compute a standard basis G'_i of J'_i for each i and combine the G'_i to G_p (a standard basis of I'_p) using chinese remaindering for polynomials. The procedure is repeated for

many primes p, where we compute the G_p in parallel until the number of primes is sufficiently large to recover the correct standard basis G' of I'. Finally, by mapping G' back to Q(t)[X], a standard basis G of I is obtained.
The procedure also works if the input is a module. For this, we consider the rings A = Q(t)[X] and A' = (Q[t]/<f>)[X]. For submodules I, I' in A^m, A'^m, respectively, we map I to I' via the map sending t to t + <f>. As above, we first choose a prime p such that f has at least two factors in characteristic p. For each factor f_{i,p} of f_p := (f mod p), we set I'_{i,p} := (I'_p mod f_{i,p}). We then compute a standard basis G'_i of I'_{i,p} over F_p[t]/<f_{i,p}> for each i and combine the G'_i to G_p (a standard basis of I'_p) using chinese remaindering for polynomials. The procedure is repeated for many primes p as described above and we finally obtain a standard basis of I.

References:

[1] E. A. Arnold: Modular algorithms for computing Groebner bases. J. Symb. Comp. 35, 403-419 (2003).

Procedures:

D.4.20.1 chinrempoly

Procedure from library `nfmodstd.lib` (see Section D.4.20 [nfmodstd_lib], page 1174).

Usage: chinrempoly(l, m); l list, m list

Return: a polynomial (resp. ideal/module) which is congruent to l[i] modulo m[i] for all i

Note: The procedure applies chinese remaindering to the first argument w.r.t. the moduli given in the second. The elements in the first list must be of the same type which can be polynomial, ideal, or module. The moduli must be of type polynomial. The elements in the second list must be distinct and co-prime.

Example:

```
LIB "nfmodstd.lib";
ring rr=97,x,dp;
poly f=x^7-7*x + 3;
ideal J=factorize(f,1);
J;
↦ J[1]=x+37
↦ J[2]=x3+9x2+20x-20
↦ J[3]=x3-46x2+17x-8
list m=J[1..ncols(J)];
list l= x^2+2*x+3, x^2+5, x^2+7;
ideal I=chinrempoly(l,m);
I;
↦ I[1]=-44x6-36x5-45x4+12x3-36x2+25x-32
ring s=0,x,dp;
list m= x^2+2*x+3, x^3+5, x^4+x^3+7;
list l=x^3 + 2, x^4 + 7, x^5 + 11;
ideal I=chinrempoly(l,m);
I;
↦ I[1]=18113/107610x8+5826/17935x7-5257/107610x6+3975/7174x5+246151/107610x\
   4+131573/53805x3-910/633x2-36239/21522x+146695/7174
int p=prime(536546513);
ring r = p, (x,y,a), (dp(2),dp(1));
poly minpolynomial = a^2+1;
```

```
ideal kf=factorize(minpolynomial,1); //return factors without multiplicity
kf;
↦ kf[1]=a+222052315
↦ kf[2]=a-222052315
ideal k=(a+1)*x2+y, 3x-ay+ a+2;
option(redSB);
ideal k1=k,kf[1];
ideal k2 =k,kf[2];
k1=std(k1);
k2=std(k2);
list l=k1,k2;
list m=kf[1..ncols(kf)];
ideal I=chinrempoly(l,m);
I=simplify(I,2);
I;
↦ I[1]=x-178848838ya+178848838a-178848837
↦ I[2]=y2-268273248ya+268273250y-4a-3
l = module(k1[2..ncols(k1)]), module(k2[2..ncols(k2)]);
module M = chinrempoly(l,m);
M;
↦ M[1]=x*gen(1)-178848838ya*gen(1)+178848838a*gen(1)-178848837*gen(1)
↦ M[2]=y2*gen(1)-268273248ya*gen(1)+268273250y*gen(1)-4a*gen(1)-3*gen(1)
```

See also: Section 5.1.8 [chinrem], page 162.

D.4.20.2 nfmodStd

Procedure from library `nfmodstd.lib` (see Section D.4.20 [nfmodstd_lib], page 1174).

Usage: nfmodStd(I, #); I ideal or module, # optional parameters

Return: standard basis of I over algebraic number field

Note: The procedure passes to Section D.4.16.1 [modStd], page 1146 if the ground field has no parameter. In this case, the optional parameters # (if given) are directly passed to Section D.4.16.1 [modStd], page 1146.

Example:

```
LIB "nfmodstd.lib";
ring r1 = (0,a),(x,y),dp;
minpoly = a^2+1;
ideal k = (a/2+1)*x^2+2/3y, 3*x-a*y+ a/7+2;
ideal I = nfmodStd(k);
I;
↦ I[1]=x+(-1/3a)*y+(1/21a+2/3)
↦ I[2]=y2+(32/5a-178/35)*y+(-4/7a-195/49)
ring rm = (0,a),(x,y),(c,dp);
minpoly = a^3+2a+7;
module M = [(a/2+1)*x^2+2/3y, 3*x-a*y+ a/7+2], [ax, y];
M = nfmodStd(M);
M;
↦ M[1]=[0,x2y+(12/5a2-24/5a+42/5)*x2+(8/5a2-6/5a+28/5)*xy+(4/15a2-8/15a+8/5\
   )*y2+(48/35a2-106/35a+24/5)*x]
↦ M[2]=[y,(3/14a2-9/28)*xy+9/2*x+(-3/2a)*y+(3/14a+3)]
↦ M[3]=[x,(-1/7a2-2/7)*y]
```

```
ring r2 = (0,a),(x,y,z),dp;
minpoly = a^3 +2;
ideal k = (a^2+a/2)*x^2+(a^2 -2/3*a)*yz, (3*a^2+1)*zx-(a+4/7)*y+ a+2/5;
ideal IJ = nfmodStd(k);
IJ;
↦ IJ[1]=xz+(138/763a2+65/763a-46/763)*y+(-96/545a2-31/545a+32/545)
↦ IJ[2]=x2+(28/45a2-14/45a+52/45)*yz
↦ IJ[3]=yz2+(-3354/23653a2-6390/23653a-7683/47306)*xy+(993/6758a2+4104/1689\
  5a+4449/33790)*x
ring r3 = 0, (x,y), dp; // ring without parameter
ideal I = x2 + y, xy - 7y + 2x;
ideal J1 = nfmodStd(I);
J1;
↦ J1[1]=y2-14x+51y
↦ J1[2]=xy+2x-7y
↦ J1[3]=x2+y
module J2 = nfmodStd(module(I));
J2;
↦ J2[1]=y2*gen(1)-14x*gen(1)+51y*gen(1)
↦ J2[2]=xy*gen(1)+2x*gen(1)-7y*gen(1)
↦ J2[3]=x2*gen(1)+y*gen(1)
ring r4 = 0, (x,y), (c,dp);
module I = [x2, x-y], [xy,0], [0,-7y + 2x];
I=nfmodStd(I);
I;
↦ I[1]=[0,x-7/2y]
↦ I[2]=[0,y2]
↦ I[3]=[xy]
↦ I[4]=[x2,5/2y]
```

See also: Section D.4.16.1 [modStd], page 1146.

D.4.21 nfmodsyz_lib

Library: nfmodsyz.lib

Purpose: Syzygy modules of submodules of free modules over algebraic number fields

Authors: D.K. Boku boku@mathematik.uni-kl.de
W. Decker decker@mathematik.uni-kl.de
C. Fieker fieker@mathematik.uni-kl.de

Overview: A library for computing the syzygy module of a given submodule I in a polynomial ring over an algebraic number field Q(t), where t is an algebraic number, using modular methods. For the case Q(t)=Q, that is, where t is an element of Q, we compute, following [1], the syzygy module of I as follows: For a submodule I of A^m with A = Q[X], we first choose a sufficiently large set of primes P and compute the reduced Groebner basis of the syzygy module of I_p, for each p in P, in parallel. We then use the Chinese remainder algorithm and rational reconstruction to obtain the syzygy module of I over Q. For the case where t is not in Q, we compute, following [2], the syzygy module of I as follows:
Let f be the minimal polynomial of t. For a submodule I in A^m with A = Q(t)[X], we map I to a submodule I' in A^m with A = (Q[t]/<f>)[X] via the map sending t to t + <f>. We first choose a prime p such that f has at least two factors in characteristic

p. For each factor f_{i,p} of f_p:= (f mod p), we set I'_{i,p} := (I'_p mod f_{i,p}). We then compute the reduced Groebner bases G'_i of the syzygy modules of I'_{i,p} over F_p[t]/<f_{i,p}> and combine the G'_i to G_p (the syzygy module of I'_p) using chinese remaindering for polynomials. As described in [2], the procedure is repeated for many primes p, where we compute the G_p in parallel until the number of primes is sufficiently large to recover the correct generating set for the syzygy module G' of I' which is, considered over Q(t), also a generating set for the syzygy module of I.

References:

[1] E. A. Arnold: Modular algorithms for computing Groebner bases. J. Symb. Comp. 35, 403-419 (2003).

[2] D. Boku, W. Decker, C. Fieker, and A. Steenpass. Groebner bases over algebraic number fields. In: Proceedings of the 2015 International Workshop on Parallel Symb. Comp. PASCO'15, pages 16-24 (2015).

Procedures:

D.4.21.1 nfmodSyz

Procedure from library `nfmodsyz.lib` (see Section D.4.21 [nfmodsyz_lib], page 1177).

Usage: nfmodSyz(I); I ideal or module

Return: syzygy module of I over an algebraic number field

Example:

```
LIB "nfmodsyz.lib";
ring r1 =(0,a),(x,y),(c,dp);
minpoly = (a^3+2a+7);
module M1 = [(a/2+1)*y, 3*x-a*y],
[y-x,y2],
[x2-xy, ax-y];
nfmodSyz(M1);
↦ _[1]=[x2y2-xy3+(a)*x2+(-a-1)*xy+y2,-3*x3+(a+3)*x2y+(-a)*xy2+(1/2a2+a)*xy+\
   (-1/2a-1)*y2,(-1/2a-1)*y3-3*x2+(a+3)*xy+(-a)*y2]
ring r2 = (0,a),(x,y,z),(dp,c);
minpoly = (a3+a+1);
module M2 = [x2z+x+(-a)*y,z2+(a+2)*x],
[y2+(a)*z+(a),(a+3)*z3+(-a)*x2],
[-xz+(a2+3)*yz,xy+(a2)*z];
nfmodSyz(M2);
↦ _[1]=x2z4*gen(3)+(3/29a2-9/29a+1/29)*x4z*gen(3)+(-1/29a2+3/29a-10/29)*x3y\
   z*gen(2)+xz4*gen(1)+(-a2-3)*yz4*gen(1)+(1/29a2-3/29a+10/29)*xy3*gen(1)+(3\
   /29a2-9/29a+1/29)*x3z*gen(1)+(-7/29a2+21/29a-12/29)*x2yz*gen(1)+(-9/29a2-\
   2/29a-3/29)*x2z2*gen(2)+(-1/29a2+3/29a-10/29)*y2z2*gen(3)+(-1/29a2+3/29a-\
   10/29)*xz3*gen(2)+xz3*gen(3)+(12/29a2-7/29a+33/29)*yz3*gen(2)+(-a)*yz3*ge\
   n(3)+(3/29a2-9/29a+1/29)*x3*gen(3)+(-1/29a2+3/29a-10/29)*x2y*gen(2)+(9/29\
   a2+2/29a+3/29)*x2y*gen(3)+(-3/29a2+9/29a-1/29)*xy2*gen(2)+(1/29a2-3/29a-1\
   9/29)*xy2*gen(3)+(1/29a2-3/29a-19/29)*x2z*gen(2)+(-3/29a2+9/29a-1/29)*xyz\
   *gen(1)+(17/29a2+7/29a+54/29)*xyz*gen(2)+(9/29a2+2/29a+3/29)*y2z*gen(1)+(\
   3/29a2-9/29a+1/29)*z3*gen(3)+(-3/29a2+9/29a-1/29)*xy*gen(1)+(-9/29a2-2/29\
   a-3/29)*xz*gen(2)+(-3/29a2-20/29a-1/29)*xz*gen(3)+(2/29a2-6/29a-9/29)*yz*\
   gen(2)+(2/29a2-6/29a-9/29)*z2*gen(1)+(3/29a2-9/29a+1/29)*z2*gen(3)+(-3/29\
   a2-20/29a-1/29)*x*gen(3)+(2/29a2-6/29a-9/29)*z*gen(1)
ring r3=0,(x,y),dp; // ring without parameter
```

```
module M3 = [x2 + y, xy], [-7y, 2x], [x2-y, 0];
nfmodSyz(M3);
↦ _[1]=x2y*gen(2)+2x2*gen(3)-2x2*gen(1)+7y2*gen(3)-y2*gen(2)+2y*gen(3)+2y*g\
   en(1)
ring r4=0,(x,y),(c,dp); // ring without parameter
module M4 = [xy, x-y],
[x2 + y, 5y],
[- 7y, 2x],
[x2-y, 0];
nfmodSyz(M4);
↦ _[1]=[0,x3-xy,-5/2x2y+5/2y2,-x3-xy-35/2y2]
↦ _[2]=[x+35/4y,-1/2x2-7/4x+7/4y,5/4xy-1/2x+1/2y,1/2x2+7/4x-7/4y]
↦ _[3]=[y2-16/1225y,-2/35x2y+156/6125x2-53/245xy+1/5y2-16/6125y,1/7xy2-78/1\
   225xy+2/49y2+8/1225y,2/35x2y-156/6125x2+53/245xy-3/35y2-296/6125y]
```

See also: Section 5.1.156 [syz], page 280.

D.4.22 noether_lib

Library: noether.lib

Purpose: Noether normalization of an ideal (not necessary homogeneous)

Authors: A. Hashemi, Amir.Hashemi@lip6.fr

Overview: A library for computing the Noether normalization of an ideal that DOES NOT require the computation of the dimension of the ideal. It checks whether an ideal is in Noether position. A modular version of these algorithms is also provided.
The procedures are based on a paper of Amir Hashemi 'Efficient Algorithms for Computing Noether Normalization' (presented in ASCM 2007)

This library computes also Castelnuovo-Mumford regularity and satiety of an ideal. A modular version of these algorithms is also provided. The procedures are based on a paper of Amir Hashemi 'Computation of Castelnuovo-Mumford regularity and satiety' (preprint 2008)

Procedures:

D.4.22.1 NPos_test

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: NPos_test (I); I monomial ideal

Return: A list whose first element is 1, if i is in Noether position, 0 otherwise. The second element of this list is a list of variables ordered such that those variables are listed first, of which a power belongs to the initial ideal of i. If i is in Noether position, the method returns furthermore the dimension of i.

Assume: i is a nonzero monomial ideal.

Example:

```
LIB "noether.lib";
ring r=0,(X,Y,a,b),dp;
poly f=X^8+a*Y^4-Y;
poly g=Y^8+b*X^4-X;
poly h=diff(f,X)*diff(g,Y)-diff(f,Y)*diff(g,X);
```

```
ideal i=f,g,h;
NPos_test(i);
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       b
↦    [2]:
↦       a
↦    [3]:
↦       Y
↦    [4]:
↦       X
```

D.4.22.2 modNpos_test

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: modNpos_test(i); i an ideal

Return: 1 if i is in Noether position 0 otherwise.

Note: This test is a probabilistic test, and it computes the initial of the ideal modulo the prime number 2147483647 (the biggest prime less than 2^31).

Example:

```
LIB "noether.lib";
ring r=0,(X,Y,a,b),dp;
poly f=X^8+a*Y^4-Y;
poly g=Y^8+b*X^4-X;
poly h=diff(f,X)*diff(g,Y)-diff(f,Y)*diff(g,X);
ideal i=f,g,h;
modNpos_test(i);
↦ // WARNING:
↦ // The procedure is probabilistic and  it computes the initial of the ide\
   al modulo the prime number 2147483647
↦ [1]:
↦    0
↦ [2]:
↦    [1]:
↦       X
↦    [2]:
↦       Y
↦    [3]:
↦       b
↦    [4]:
↦       a
```

D.4.22.3 NPos

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: NPos(i); i ideal

Return: A linear map phi such that phi(i) is in Noether position

Example:

```
LIB "noether.lib";
ring r=0,(X,Y,a,b),dp;
poly f=X^8+a*Y^4-Y;
poly g=Y^8+b*X^4-X;
poly h=diff(f,X)*diff(g,Y)-diff(f,Y)*diff(g,X);
ideal i=f,g,h;
NPos(i);
↦ The dimension of the ideal is:
↦ 1
↦ _[1]=X
↦ _[2]=Y
↦ _[3]=a
↦ _[4]=53X+27Y-75a+b and the time of this computation is: 693 /10 sec.
```

D.4.22.4 modNPos

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: modNPos(i); i ideal

Return: A linear map phi such that phi(i) is in Noether position

Note: It uses the procedure modNpos_test to test Noether position.

Example:

```
LIB "noether.lib";
ring r=0,(X,Y,a,b),dp;
poly f=X^8+a*Y^4-Y;
poly g=Y^8+b*X^4-X;
poly h=diff(f,X)*diff(g,Y)-diff(f,Y)*diff(g,X);
ideal i=f,g,h;
modNPos(i);
↦ // WARNING:
↦ // The procedure is probabilistic and  it computes the initial of the ide\
   al modulo the prime number 2147483647
↦ // WARNING:
↦ // The procedure is probabilistic and  it computes the initial of the ide\
   al modulo the prime number 2147483647
↦ // WARNING:
↦ // The procedure is probabilistic and  it computes the initial of the ide\
   al modulo the prime number 2147483647
↦ The dimension of the ideal is:
↦ 1
↦ _[1]=X
↦ _[2]=Y
↦ _[3]=a
↦ _[4]=53X+27Y-75a+b and the time of this computation is: 10 /10 sec.
```

D.4.22.5 nsatiety

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: nsatiety (i); i ideal,

Return: an integer, the satiety of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal of the basering R=K[x(0)..x(n)].

Theory: The satiety, or saturation index, of a homogeneous ideal i is the least integer s such that, for all d>=s, the degree d part of the ideals i and isat=sat(i,maxideal(1)) coincide.

Example:

```
LIB "noether.lib";
ring r=0,(t,a,b,c,d),dp;
ideal i=b4-a3d, ab3-a3c, bc4-ac3d-bcd3+ad4, c6-bc3d2-c3d3+bd5, ac5-b2c3d-ac2d3+b2d4,
nsatiety(i);
↦ sat(i)=0 and the time of this computation: 2/100sec.
```

D.4.22.6 modsatiety

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: modsatiety(i); i ideal,

Return: an integer, the satiety of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal of the basering R=K[x(0)..x(n)].

Theory: The satiety, or saturation index, of a homogeneous ideal i is the least integer s such that, for all d>=s, the degree d part of the ideals i and isat=sat(i,maxideal(1)) coincide.

Note: This is a probabilistic procedure, and it computes the initial of the ideal modulo the prime number 2147483647 (the biggest prime less than 2^31).

Example:

```
LIB "noether.lib";
ring r=0,(t,a,b,c,d),dp;
ideal i=b4-a3d, ab3-a3c, bc4-ac3d-bcd3+ad4, c6-bc3d2-c3d3+bd5, ac5-b2c3d-ac2d3+b2d4,
modsatiety(i);
↦ // WARNING: The characteristic of base field must be zero.
↦ // The procedure is probabilistic and  it computes the
↦ //initial ideals modulo the prime number 2147483647.
↦ msat(i)=0 and the time of this computation: 2/100sec.
```

D.4.22.7 regCM

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: regCM (i); i ideal

Return: the Castelnuovo-Mumford regularity of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal.

Example:

```
LIB "noether.lib";
ring r=0,(t,a,b,c,d),dp;
ideal i=b4-a3d, ab3-a3c, bc4-ac3d-bcd3+ad4, c6-bc3d2-c3d3+bd5, ac5-b2c3d-ac2d3+b2d4,
regCM(i);
↦ reg(i)=7 and the time of this computation: 4 sec./100
```

D.4.22.8 modregCM

Procedure from library `noether.lib` (see Section D.4.22 [noether_lib], page 1179).

Usage: modregCM(i); i ideal

Return: an integer, the Castelnuovo-Mumford regularity of i.
(returns -1 if i is not homogeneous)

Assume: i is a homogeneous ideal and the characteristic of base field is zero..

Note: This is a probabilistic procedure, and it computes the initial of the ideal modulo the prime number 2147483647 (the biggest prime less than 2^31).

Example:

```
LIB "noether.lib";
ring r=0,(t,a,b,c,d),dp;
ideal i=b4-a3d, ab3-a3c, bc4-ac3d-bcd3+ad4, c6-bc3d2-c3d3+bd5, ac5-b2c3d-ac2d3+b2d4,
modregCM(i);
↦ // WARNING: The characteristic of base field must be zero.
↦ // This procedure is probabilistic and  it computes the initial
↦ //ideals modulo the prime number 2147483647
↦ // ** redefining zz (  list l11;int zz;) noether.lib::modregCM:937
↦ mreg(i)=7 and the time of this computation: 4sec./100
```

D.4.23 normal_lib

Library: normal.lib

Purpose: Normalization of Affine Rings

Authors: J. Boehm boehm@mathematik.uni-kl.de,
W. Decker decker@mathematik.uni-kl.de,
G.-M. Greuel, greuel@mathematik.uni-kl.de,
S. Laplagne, slaplagn@dm.uba.ar,
G. Pfister, pfister@mathematik.uni-kl.de,
A. Steenpass steenpass@mathematik.uni-kl.de,
S. Steidel steidel@mathematik.uni-kl.de,
P. Chini, chini@rhrk.uni-kl.de (normalConductor)

Procedures: See also: Section D.4.12 [integralbasis_lib], page 1114.

D.4.23.1 normal

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: normal(id [,choose]); id = radical ideal, choose = list of options.
Optional parameters in list choose (can be entered in any order):
Decomposition:
- "equidim" -> computes first an equidimensional decomposition of the input ideal, and then the normalization of each component (default).
- "prim" -> computes first the minimal associated primes of the input ideal, and then the normalization of each prime. (When the input ideal is not prime and the minimal associated primes are easy to compute, this method is usually faster than "equidim".)
- "noDeco" -> no preliminary decomposition is done. If the ideal is not equidimensional radical, output might be wrong.

- "isPrim" -> assumes that the ideal is prime. If this assumption does not hold, the output might be wrong.
- "noFac" -> factorization is avoided in the computation of the minimal associated primes;
Other:
- "useRing" -> uses the original ring ordering.
If this option is set and if the ring ordering is not global, normal will change to a global ordering only for computing radicals and prime or equidimensional decompositions.
If this option is not set, normal changes to dp ordering and performs all computations with respect to this ordering.
- "withDelta" (or "wd") -> returns also the delta invariants. Not valid for options "locNormal" and "modNormal".
If the optional parameter choose is not given or empty, only "equidim" but no other option is used.
- list("inputJ", ideal inputJ) -> takes as initial test ideal the ideal inputJ. This option is only for use in other procedures. Using this option, the result might not be the normalization.
(Option only valid for global algorithm.)
- list("inputC", ideal inputC) -> takes as initial conductor the ideal inputC. This option is only for use in other procedures. Using this option, the result might not be the normalization.
(Option only valid for global algorithm.)
Options used for computing integral basis (over rings of two variables):
- "normalCheck" -> checks only if the ring is normal. The output is 1 if the ring is normal and 0 if not.
- "var1" -> uses a polynomial in the first variable as universal denominator.
- "var2" -> uses a polynomial in the second variable as universal denominator.
- "GBRadical" -> it computes a Groebner basis of the radical ideal computed for the test ideal.
- "noGBRadical" -> it does not compute a Groebner basis of the radical ideal computed for the test ideal.
- "locNormal" -> uses locNormal procedure to compute the normalization, which implements a local-to-global algorithm (for this option it is required that the ideal I is prime).
- "modular" -> combined with "locNormal" option, it will use modNormal algorithm for the local computations.
- "modNormal" -> uses modNormal procedure to compute the normalization, which implements a modular algorithm (for this option it is required that the ideal I is prime).
- list("nPrimes", int n) -> parameter for modNormal. We consider n primes at a time until the result lifted to the rationals is correct modulo one additional prime. (default: n = 10)
- "noVerification" -> if the modular approach is used (using modNormal option or modular option in locNormal), the
result will not be verified.
If the optional parameter choose is not given or empty, only "equidim" but no other option is used.
- "withRing" -> the ring structure of the normalization is computed. The number of variables in the new ring is reduced as much as possible. This is the default option, except when using modNormal or locNormal options.

Assume: The ideal must be radical, for non-radical ideals the output may be wrong (id=radical(id); makes id radical). However, when using the "prim" option the minimal associated primes of id are computed first and hence normal computes the normalization of the radical of id.

Note: "isPrim" should only be used if id is known to be prime.

Return: If "withRing" option is set, a list, say nor, of size 2 (resp. 3 with option "withDelta").
Let R denote the basering and id the input ideal.
* nor[1] is a list of r rings, where r is the number of associated primes P_i with option "prim" (resp. >= no of equidimenensional components P_i with option "equidim").
Each ring Ri := nor[1][i], i=1..r, contains two ideals with given names `norid` and `normap` such that:
- Ri/norid is the normalization of the i-th component, i.e. the integral closure of R/P_i in its field of fractions (as affine ring); - `normap` gives the normalization map from R/id to Ri/norid for each i.
- the direct sum of the rings Ri/norid, i=1,..r, is the normalization of R/id as affine algebra;
* nor[2] is a list of size r with information on the normalization of the i-th component as module over the basering R:
nor[2][i] is an ideal, say U, in R such that the integral closure of basering/P_i is generated as module over R by 1/c * U, with c the last element U[size(U)] of U.
* nor[3] (if option "withDelta" is set) is a list of an intvec of size r, the delta invariants of the r components, and an integer, the total delta invariant of basering/id (-1 means infinite, and 0 that R/P_i resp. R/id is normal).
If "withRing" option is not set, a list of an ideal U and a polynomial d such that U/d is the normalization

Theory: We use here a general algorithm described in [G.-M.Greuel, S.Laplagne, F.Seelisch: Normalization of Rings (2009)].
The procedure computes the R-module structure, the algebra structure and the delta invariant of the normalization of R/id:
The normalization of R/id is the integral closure of R/id in its total ring of fractions. It is a finitely generated R-module and nor[2] computes R-module generators of it. More precisely: If U:=nor[2][i] and c:=U[size(U)], then c is a non-zero divisor and U/c is an R-module in the total ring of fractions, the integral closure of R/P_i. Since U[size(U)]/c is equal to 1, R/P_i resp. R/id is contained in the integral closure.
The normalization is also an affine algebra over the ground field and nor[1] presents it as such. For geometric considerations nor[1] is relevant since the variety of the ideal norid in Ri is the normalization of the variety of the ideal P_i in R.
The delta invariant of a reduced ring A is dim_K(normalization(A)/A). For A=K[x1,...,xn]/id we call this number also the delta invariant of id. nor[3] returns the delta invariants of the components P_i and of id.

Note: To use the i-th ring type e.g.: `def R=nor[1][i]; setring R;`.
Increasing/decreasing printlevel displays more/less comments (default: printlevel=0).
Implementation works also for local rings.
Not implemented for quotient rings.
If the input ideal id is weighted homogeneous a weighted ordering may be used together with the useRing-option (qhweight(id); computes weights).

Example:

```
LIB "normal.lib";
printlevel = printlevel+1;
ring s = 0,(x,y),dp;
ideal i = (x2-y3)*(x2+y2)*x;
list nor = normal(i, "withDelta", "prim");
↦ // Computing the minimal associated primes...
↦ [1]:
↦    _[1]=-y3+x2
↦ [2]:
↦    _[1]=x2+y2
↦ [3]:
↦    _[1]=x
↦ 
↦ // number of components is 3
↦ 
↦ // start computation of component 1
↦    --------------------------------
↦ Computing the jacobian ideal...
↦ 
↦ The universal denominator is  x
↦ The original singular locus is
↦ _[1]=x
↦ _[2]=y2
↦ 
↦ The radical of the original singular locus is
↦ J[1]=x
↦ J[2]=y
↦ The non zero divisor is  y
↦ 
↦ Preliminar step begins.
↦ Computing the quotient (DJ : J)...
↦ In this step, we have the ring 1/c * U, with c = y
↦ and U =
↦ U[1]=y
↦ U[2]=x
↦ 
↦ Step  1  begins.
↦ Computing the test ideal...
↦ Computing the quotient (c*D*cJ : cJ)...
↦ The ring in the previous step was already normal.
↦ 
↦ // start computation of component 2
↦    --------------------------------
↦ Computing the jacobian ideal...
↦ 
↦ The universal denominator is  y
↦ The original singular locus is
↦ _[1]=y
↦ _[2]=x
↦ 
↦ The radical of the original singular locus is
↦ J[1]=x
↦ J[2]=y
↦ The non zero divisor is  y
↦ 
↦ Preliminar step begins.
↦ Computing the quotient (DJ : J)...
↦ In this step, we have the ring 1/c * U, with c = y
↦ and U =
↦ U[1]=y
↦ U[2]=x
↦ 
↦ Step  1  begins.
↦ Computing the test ideal...
↦ Computing the quotient (c*D*cJ : cJ)...
↦ The ring in the previous step was already normal.
↦ 
↦ // start computation of component 3
↦    -------------------------------
↦ Computing the jacobian ideal...
↦ // Sum of delta for all components:  2
↦ // Computing the sum of the intersection multiplicities of the components\
   ...
↦ // Intersection multiplicity is :  11
↦ 
↦ // 'normal' created a list, say nor, of three elements.
↦ // To see the list type
↦       nor;
↦ 
↦ // * nor[1] is a list of 3 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦ 
↦ // * nor[2] is a list of 3 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
↦ 
↦ // * nor[3] is a list of an intvec of size 3 the delta invariants
↦ // of the components, and an integer, the total delta invariant
↦ // of R/id (-1 means infinite, and 0 that R/P_i resp. R/id is normal).
nor;
↦ [1]:
↦    [1]:
↦       // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering dp
↦ //                  : names    x y
↦ //        block   3 : ordering C
↦    [2]:
↦       // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering dp
↦ //                  : names    x y
↦ //        block   3 : ordering C
↦    [3]:
↦       // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x y
↦ //        block   2 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=x
↦       _[2]=y
↦    [2]:
↦       _[1]=x
↦       _[2]=y
↦    [3]:
↦       _[1]=1
↦ [3]:
↦    [1]:
↦       1,1,0
↦    [2]:
↦       13
// 2 branches have delta = 1, and 1 branch has delta = 0
// the total delta invariant is 13
def R2 = nor[1][2];  setring R2;
norid; normap;
↦ norid[1]=-T(1)*y+x
↦ norid[2]=T(1)*x+y
↦ norid[3]=T(1)^2+1
↦ norid[4]=x^2+y^2
↦ normap[1]=x
↦ normap[2]=y
printlevel = printlevel-1;
ring r = 2,(x,y,z),dp;
ideal i = z3-xy4;
list nor = normal(i, "withDelta", "prim");  nor;
↦ 
↦ // 'normal' created a list, say nor, of three elements.
↦ // To see the list type
↦       nor;
↦ 
↦ // * nor[1] is a list of 1 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦
↦ // * nor[2] is a list of 1 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
↦
↦ // * nor[3] is a list of an intvec of size 1 the delta invariants
↦ // of the components, and an integer, the total delta invariant
↦ // of R/id (-1 means infinite, and 0 that R/P_i resp. R/id is normal).
↦ [1]:
↦    [1]:
↦       // coefficients: ZZ/2
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2)
↦ //        block   2 : ordering dp
↦ //                  : names    x y z
↦ //        block   3 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=xy2z
↦       _[2]=xy3
↦       _[3]=z2
↦ [3]:
↦    [1]:
↦       -1
↦    [2]:
↦       -1
// the delta invariant is infinite
// xy2z/z2 and xy3/z2 generate the integral closure of r/i as r/i-module
// in its quotient field Quot(r/i)
// the normalization as affine algebra over the ground field:
def R = nor[1][1]; setring R;
norid; normap;
↦ norid[1]=T(1)*y+T(2)*z
↦ norid[2]=T(2)*y+z
↦ norid[3]=T(1)*z+x*y^2
↦ norid[4]=T(1)^2+x*z
↦ norid[5]=T(1)*T(2)+x*y
↦ norid[6]=T(2)^2+T(1)
↦ norid[7]=x*y^4+z^3
↦ normap[1]=x
↦ normap[2]=y
↦ normap[3]=z
```

See also: Section D.4.23.5 [normalC], page 1194; Section D.4.23.4 [normalP], page 1191.

D.4.23.2 locNormal

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: locNormal(I [,options]); I = prime ideal, options = list of options.
Optional parameters in list options (can be entered in any order):
modular: use a modular approach for the local computations. The number of primes is increased one at a time, starting with 2 primes, until the result stabilizes.
noVerification: if the modular approach is used, the result will not be verified.

Assume: I is a prime ideal (the algorithm will also work for radical ideals as long as the normal command does not detect that the ideal under consideration is not prime).

Return: a list of an ideal U and a universal denominator d such that U/d is the normalization.

Remarks: We use the local-to-global algorithm given in [1] to compute the normalization of A = R/I, where R is the basering.
The idea is to stratify the singular locus of A, apply the normalization algorithm given in [2] locally at each stratum, and put the local results together.
If the option modular is given, the result is returned as a probabilistic result or verified, depending on whether the option noVerification is used or not.
The normalization of A is represented as an R-module by returning a list of U and d, where U is an ideal of A and d is an element of A such that U/d is the normalization of A. In fact, U and d are returned as an ideal and a polynomial of the base ring R.

References:
[1] Janko Boehm, Wolfram Decker, Santiago Laplagne, Gerhard Pfister, Stefan Steidel, Andreas Steenpass: Parallel algorithms for normalization, http://arxiv.org/abs/1110.4299, 2011.
[2] Gert-Martin Greuel, Santiago Laplagne, Frank Seelisch: Normalization of Rings, Journal of Symbolic Computation 9 (2010), p. 887-901

Example:

```
     LIB "normal.lib";
     ring R = 0,(x,y,z),dp;
     int k = 4;
     poly f = (x^(k+1)+y^(k+1)+z^(k+1))^2-4*(x^(k+1)*y^(k+1)+y^(k+1)*z^(k+1)+z^(k+1)*x^(k+
     f = subst(f,z,3x-2y+1);
     ring S = 0,(x,y),dp;
     poly f = imap(R,f);
     ideal i = f;
     list L = locNormal(i);
```

See also: Section D.4.23 [normal_lib], page 1183.

D.4.23.3 modNormal

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: modNormal(I, n [,options]); I = prime ideal, n = positive integer, options = list of options.
Optional parameters in list options (can be entered in any order):
noVerification: do not verify the result.
printTimings: print timings.
int ncores: number of cores to be used (default = 1).

Assume: I is a prime ideal (the algorithm will also work for radical ideals as long as the normal command does not detect that the ideal under consideration is not prime).

Return: a list of an ideal U and a universal denominator d such that U/d is the normalization.

Remarks: We use the algorithm given in [1] to compute the normalization of A = R/I where R is the basering. We apply the algorithm for n primes at a time until the result lifted to the rationals is correct modulo one additional prime. Depending on whether the option noVerification is used or not, the result is returned as a probabilistic result or verified over the rationals.

The normalization of A is represented as an R-module by returning a list of U and d, where U is an ideal of A and d is an element of A such that U/d is the normalization of A. In fact, U and d are returned as an ideal and a polynomial of the base ring R.

Example:

```
LIB "normal.lib";
ring R = 0,(x,y,z),dp;
int k = 4;
poly f = (x^(k+1)+y^(k+1)+z^(k+1))^2-4*(x^(k+1)*y^(k+1)+y^(k+1)*z^(k+1)+z^(k+1)*x^(k
f = subst(f,z,3x-2y+1);
ring S = 0,(x,y),dp;
poly f = imap(R,f);
ideal i = f;
list L = modNormal(i,1,"noVerification");
```

See also: Section D.4.12 [integralbasis_lib], page 1114.

D.4.23.4 normalP

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: normalP(id [,choose]); id = radical ideal, choose = optional list of strings.
Optional parameters in list choose (can be entered in any order):
"withRing", "isPrim", "noFac", "noRed", where
- "noFac" -> factorization is avoided during the computation of the minimal associated primes.
- "isPrim" -> assumes that the ideal is prime. If the assumption does not hold, output might be wrong.
- "withRing" -> the ring structure of the normalization is computed. The number of variables in the new ring is reduced as much as possible.
- "noRed" -> when computing the ring structure, no reduction on the number of variables is done, it creates one new variable for every new module generator of the integral closure in the quotient field.

Assume: The characteristic of the ground field must be positive. If the option "isPrim" is not set, the minimal associated primes of id are computed first and hence normalP computes the normalization of the radical of id. If option "isPrim" is set, the ideal must be a prime ideal otherwise the result may be wrong.

Return: a list, say 'nor' of size 2 (resp. 3 if "withRing" is set).
** If option "withRing" is not set:
Only the module structure is computed:
* nor[1] is a list of ideals Ii, i=1..r, in the basering R where r is the number of minimal associated prime ideals P_i of the input ideal id, describing the module structure:
If Ii is given by polynomials g_1,...,g_k in R, then c:=g_k is non-zero in the ring R/P_i and g_1/c,...,g_k/c generate the integral closure of R/P_i as R-module in the quotient field of R/P_i.
* nor[2] shows the delta invariants: it is a list of an intvec of size r, the delta invariants

of the r components, and an integer, the total delta invariant of R/id
(-1 means infinite, and 0 that R/P_i resp. R/id is normal).
** If option "withRing" is set:
The ring structure is also computed, and in this case:
* nor[1] is a list of r rings.
Each ring Ri = nor[1][i], i=1..r, contains two ideals with given names `norid` and `normap` such that
- Ri/norid is the normalization of R/P_i, i.e. isomorphic as K-algebra (K the ground field) to the integral closure of R/P_i in the field of fractions of R/P_i;
- the direct sum of the rings Ri/norid is the normalization of R/id;
- `normap` gives the normalization map from R to Ri/norid.
* nor[2] gives the module generators of the normalization of R/P_i, it is the same as nor[1] if "withRing" is not set.
* nor[3] shows the delta invariants, it is the same as nor[2] if "withRing" is not set.

Theory: normalP uses the Leonard-Pellikaan-Singh-Swanson algorithm (using the Frobenius) cf. [A. K. Singh, I. Swanson: An algorithm for computing the integral closure, arXiv:0901.0871].
The delta invariant of a reduced ring A is dim_K(normalization(A)/A). For A=K[x1,...,xn]/id we call this number also the delta invariant of id. The procedure returns the delta invariants of the components P_i and of id.

Note: To use the i-th ring type: `def R=nor[1][i]; setring R;`.
Increasing/decreasing printlevel displays more/less comments (default: printlevel = 0).
Not implemented for local or mixed orderings or quotient rings. For local or mixed orderings use proc 'normal'.
If the input ideal id is weighted homogeneous a weighted ordering may be used (qhweight(id); computes weights).
Works only in characteristic p > 0; use proc normal in char 0.

Example:

```
LIB "normal.lib";
ring r  = 11,(x,y,z),wp(2,1,2);
ideal i = x*(z3 - xy4 + x2);
list nor= normalP(i); nor;
↦
↦ // 'normalP' computed a list, say nor, of two lists:
↦ // To see the result, type
↦      nor;
↦
↦ // * nor[1] is a list of 2 ideal(s), where each ideal nor[1][i] consists
↦ // of elements g1..gk of the basering R such that gj/gk generate the inte\
   gral
↦ // closure of R/P_i (where P_i is a min. associated prime of the input id\
   eal)
↦ // as R-module in the quotient field of R/P_i;
↦
↦ // * nor[2] shows the delta-invariant of each component and of the input \
   ideal
↦ // (-1 means infinite, and 0 that R/P_i is normal).
↦ [1]:
↦    [1]:
```

```
↦       _[1]=1
↦    [2]:
↦       _[1]=1
↦ [2]:
↦    [1]:
↦       0,0
↦    [2]:
↦       -1
//the result says that both components of i are normal, but i itself
//has infinite delta
ring s = 2,(x,y),dp;
ideal i = y*((x-y^2)^2 - x^3);
list nor = normalP(i,"withRing"); nor;
↦
↦ // 'normalP' created a list, say nor, of three lists:
↦ // To see the result, type
↦      nor;
↦
↦ // * nor[1] is a list of 2 ring(s):
↦ // To access the i-th ring nor[1][i] give it a name, say Ri, and type e.g\
   .
↦      def R1 = nor[1][1]; setring R1;  norid; normap;
↦ // for the other rings type first setring R; (if R is the name of your
↦ // original basering) and then continue as for R1;
↦ // Ri/norid is the affine algebra of the normalization of the i-th
↦ // component R/P_i (where P_i is a min. associated prime of the input ide\
   al)
↦ // and normap the normalization map from R to Ri/norid;
↦
↦ // * nor[2] is a list of 2 ideal(s), each ideal nor[2][i] consists of
↦ // elements g1..gk of r such that the gj/gk generate the integral
↦ // closure of R/P_i as R-module in the quotient field of R/P_i.
↦
↦ // * nor[3] shows the delta-invariant of each component and of the input
↦ // ideal (-1 means infinite, and 0 that r/P_i is normal).
↦ [1]:
↦    [1]:
↦       // coefficients: ZZ/2
↦ // number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    x
↦ //        block   2 : ordering C
↦    [2]:
↦       // coefficients: ZZ/2
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering dp
↦ //                  : names    y
↦ //        block   3 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       _[1]=y3+xy
↦       _[2]=x2
↦ [3]:
↦    [1]:
↦       0,3
↦    [2]:
↦       6
def R2  = nor[1][2]; setring R2;
norid; normap;
↦ norid[1]=T(1)^4+T(1)^3*y+T(1)^2+y^2
↦ norid[2]=T(1)^3+T(1)+y
↦ norid[3]=T(1)^6+T(1)^5*y+T(1)^4*y^2+T(1)^3*y^3+T(1)^2+T(1)*y+y^4
↦ normap[1]=T(1)^2+T(1)*y+1
↦ normap[2]=y
```

See also: Section D.4.23.1 [normal], page 1183; Section D.4.23.5 [normalC], page 1194.

D.4.23.5 normalC

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: normalC(id [,choose]); id = radical ideal, choose = optional list of string.
Optional parameters in list choose (can be entered in any order):
Decomposition:
- "equidim" -> computes first an equidimensional decomposition, and then the normalization of each component (default).
- "prim" -> computes first the minimal associated primes, and then the normalization of each prime.
- "noDeco" -> no preliminary decomposition is done. If the ideal is not equidimensional radical, output might be wrong.
- "isPrim" -> assumes that the ideal is prime. If the assumption does not hold, output might be wrong.
- "noFac" -> factorization is avoided in the computation of the minimal associated primes;
Other:
- "withGens" -> the minimal associated primes P_i of id are computed and for each P_i, algebra generators of the integral closure of basering/P_i are computed as elements of its quotient field;
If choose is not given or empty, the default options are used.

Assume: The ideal must be radical, for non-radical ideals the output may be wrong (id=radical(id); makes id radical). However, if option "prim" is set the minimal associated primes are computed first and hence normalC computes the normalization of the radical of id. "isPrim" should only be used if id is known to be irreducible.

Return: a list, say nor, of size 2 (resp. 3 if option "withGens" is set).
* nor[1] is always a of r rings, where r is the number of associated primes with option "prim" (resp. >= no of equidimenensional components with option "equidim").
Each ring Ri=nor[1][i], i=1..r, contains two ideals with given names `norid` and `normap` such that
- Ri/norid is the normalization of the i-th component, i.e. the integral closure in its field of fractions as affine ring, i.e. Ri is given in the form K[X(1..p),T(1..q)], where K

is the ground field; - normap gives the normalization map from basering/id to Ri/norid for each i (the j-th element of normap is mapped to the j-th variable of R).
- the direct sum of the rings Ri/norid is the normalization of basering/id;
** If option "withGens" is not set:
* nor[2] shows the delta invariants: nor[2] is a list of an intvec of size r, the delta invariants of the r components, and an integer, the delta invariant of basering/id. (-1 means infinite, 0 that basering/P_i resp. basering/input is normal, -2 means that delta resp. delta of one of the components is not computed (which may happen if "equidim" is given).
** If option "withGens" is set:
* nor[2] is a list of ideals Ii=nor[2][i], i=1..r, in the basering, generating the integral closure of basering/P_i in its quotient field as K-algebra (K the ground field):
If Ii is given by polynomials g_1,...,g_k, then c:=g_k is a non-zero divisor and the j-th variables of the ring Ri satisfies var(j)=g_j/c, j=1..k-1, as element in the quotient field of basering/P_i. The g_j/g_k+1 are K-algebra generators of the integral closure of basering/P_i.
* nor[3] shows the delta invariant as above.

Theory: We use the Grauert-Remmert-de Jong algorithm [c.f. G.-M. Greuel, G. Pfister: A SINGULAR Introduction to Commutative Algebra, 2nd Edition. Springer Verlag (2007)].
The procedure computes the algebra structure and the delta invariant of the normalization of R/id:
The normalization is an affine algebra over the ground field K and nor[1] presents it as such: Ri = K[X(1..p),T(1..q)] and Ri/norid is the integral closure of R/P_i; if option "withGens" is set the X(j) and T(j) are expressed as quotients in the total ring of fractions. Note that the X(j) and T(j) generate the integral closure as K-algebra, but not necessarily as R-module (since relations of the form X(1)=T(1)*T(2) may have been eliminated). Geometrically the algebra structure is relevant since the variety of the ideal norid in Ri is the normalization of the variety of the ideal P_i in R.
The delta invariant of a reduced ring A is dim_K(normalization(A)/A). For A=K[x1,...,xn]/id we call this number also the delta invariant of id. nor[3] returns the delta invariants of the components P_i and of id.

Note: To use the i-th ring type: `def R=nor[1][i]; setring R;`.
Increasing/decreasing printlevel displays more/less comments (default: printlevel=0).
Not implemented for local or mixed orderings or quotient rings. For local or mixed orderings use proc 'normal'.
If the input ideal id is weighted homogeneous a weighted ordering may be used (qhweight(id); computes weights).

Example:

```
LIB "normal.lib";
printlevel = printlevel+1;
ring s = 0,(x,y),dp;
ideal i = (x2-y3)*(x2+y2)*x;
list nor = normalC(i);
↦ // We use method 'prim'
↦
↦ // number of irreducible components: 3
↦
↦ // computing the normalization of component 1
↦    ---------------------------------------
↦ // delta of component 1
↦ 1
↦ 
↦ // computing the normalization of component 2
↦    ---------------------------------------
↦ // delta of component 2
↦ 1
↦ 
↦ // computing the normalization of component 3
↦    ---------------------------------------
↦ // delta of component 3
↦ 0
↦ // Sum of delta for all components
↦ 2
↦ // Compute intersection multiplicities of the components
↦ 
↦ // 'normalC' created a list, say nor, of two lists:
↦ // To see the result, type
↦       nor;
↦ 
↦ // * nor[1] is a list of 3 ring(s).
↦ // To access the i-th ring nor[1][i] give it a name, say Ri, and type e.g\
   .
↦       def R1 = nor[1][1];  setring R1;  norid;  normap;
↦ // and similair for the other rings nor[1][i];
↦ // Ri/norid is the affine algebra of the normalization of r/P_i  (where P\
   _i
↦ // is an associated prime or an equidimensional part of the input ideal i\
   d)
↦ // and normap the normalization map from the basering to Ri/norid;
↦ 
↦ // * nor[2] shows the delta-invariant of each component and of id
↦ // (-1 means infinite, 0 that r/P_i resp. r/id is normal, and -2 that del\
   ta
↦ // of a component was not computed).
nor;
↦ [1]:
↦    [1]:
↦       // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering C
↦    [2]:
↦       // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2)
↦ //        block   2 : ordering C
↦    [3]:
↦       // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering C
↦ [2]:
↦    [1]:
↦       0,1,1
↦    [2]:
↦       13
// 2 branches have delta = 1, and 1 branch has delta = 0
// the total delta invariant is 13
def R2 = nor[1][2];  setring R2;
norid; normap;
↦ norid[1]=T(2)^2+1
↦ normap[1]=-T(1)*T(2)
↦ normap[2]=T(1)
ring r = 2,(x,y,z),dp;
ideal i = z3-xy4;
nor = normalC(i);  nor;
↦ // We use method 'equidim'
↦ 
↦ // number of equidimensional components: 1
↦ 
↦ // number of components after application of facstd: 1
↦ 
↦ // computing the normalization of component 1
↦    ----------------------------------------
↦ // delta of component 1
↦ -1
↦ // Sum of delta for all components
↦ -1
↦ 
↦ // 'normalC' created a list, say nor, of two lists:
↦ // To see the result, type
↦       nor;
↦ 
↦ // * nor[1] is a list of 1 ring(s).
↦ // To access the i-th ring nor[1][i] give it a name, say Ri, and type e.g\
  .
↦       def R1 = nor[1][1];  setring R1;  norid;  normap;
↦ // and similair for the other rings nor[1][i];
↦ // Ri/norid is the affine algebra of the normalization of r/P_i  (where P\
  _i
↦ // is an associated prime or an equidimensional part of the input ideal i\
  d)
↦ // and normap the normalization map from the basering to Ri/norid;
↦ 
↦ // * nor[2] shows the delta-invariant of each component and of id
↦ // (-1 means infinite, 0 that r/P_i resp. r/id is normal, and -2 that del\
  ta
↦ // of a component was not computed).
↦ [1]:
↦    [1]:
↦       // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3)
↦ //        block   2 : ordering C
↦ [2]:
↦    [1]:
↦       -1
↦    [2]:
↦       -1
// the delta invariant is infinite
// xy2z/z2 and xy3/z2 generate the integral closure of r/i as r/i-module
// in its quotient field Quot(r/i)
// the normalization as affine algebra over the ground field:
def R = nor[1][1]; setring R;
norid; normap;
↦ norid[1]=T(3)^3+T(1)*T(2)
↦ normap[1]=T(1)
↦ normap[2]=T(2)
↦ normap[3]=T(2)*T(3)
setring r;
nor = normalC(i, "withGens", "prim");    // a different algorithm
↦ 
↦ // We use method 'withGens'
↦ // Computing minimal associated primes...
↦ // number of irreducible components is 1
↦ 
↦ pause>
↦ // Computing normalization of component 1
↦    ---------------------------------------
↦ 
↦ // 'normalC' created a list, say nor, of three lists:
↦ // To see the list type
↦       nor;
↦ 
↦ // * nor[1] is a list of 1 ring(s)
↦ // To access the i-th ring nor[1][i] give it a name, say Ri, and type e.g\
   .
↦      def R1 = nor[1][1]; setring R1;  norid; normap;
↦ // For the other rings type first (if R is the name of your original base\
   ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of the i-th
↦ // component R/P_i (where P_i is an associated prime of the input ideal i\
   d)
↦ // and normap the normalization map from R to Ri/norid.
↦ 
↦ // * nor[2] is a list of 1 ideal(s), each ideal nor[2][i] consists of
↦ // elements g1..gk of R such that the gj/gk generate the integral
↦ // closure of R/P_i as sub-algebra in the quotient field of R/P_i, with
↦ // gj/gk being mapped by normap to the j-th variable of Ri;
↦ 
↦ // * nor[3] shows the delta-invariant of each component and of id
↦ // (-1 means infinite, and 0 that R/P_i resp. R/id is normal).
nor;
↦ [1]:
↦    [1]:
↦       // coefficients: ZZ/2
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) T(2) T(3)
↦ //        block   2 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=x2y3
↦       _[2]=z3
↦       _[3]=xy3z
↦       _[4]=xy2z2
↦       _[5]=xyz2
↦       _[6]=xy2z
↦       _[7]=xy3
↦ [3]:
↦    [1]:
↦       -1
↦    [2]:
↦       -1
```

See also: Section D.4.23.1 [normal], page 1183; Section D.4.23.4 [normalP], page 1191.

D.4.23.6 HomJJ

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: HomJJ (Li); Li = list: ideal SBid, ideal id, ideal J, poly p

Assume: R = P/id, P = basering, a polynomial ring, id an ideal of P,
SBid = standard basis of id,
J = ideal of P containing the polynomial p,
p = nonzero divisor of R

Compute: Endomorphism ring End_R(J)=Hom_R(J,J) with its ring structure as affine ring, together with the map R –> Hom_R(J,J) of affine rings, where R is the quotient ring of P modulo the standard basis SBid.

Return: a list l of three objects

l[1] : a polynomial ring, containing two ideals, 'endid' and 'endphi'
such that l[1]/endid = Hom_R(J,J) and
endphi describes the canonical map R -> Hom_R(J,J)
l[2] : an integer which is 1 if phi is an isomorphism, 0 if not
l[3] : an integer, = dim_K(Hom_R(J,J)/R) (the contribution to delta)
if the dimension is finite, -1 otherwise

Note: printlevel >=1: display comments (default: printlevel=0)

Example:

```
LIB "normal.lib";
ring r   = 0,(x,y),wp(2,3);
ideal id = y^2-x^3;
ideal J  = x,y;
```

```
poly p   = x;
list Li  = std(id),id,J,p;
list L   = HomJJ(Li);
def end = L[1];    // defines ring L[1], containing ideals endid, endphi
setring end;       // makes end the basering
end;
↦ // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering C
endid;             // end/endid is isomorphic to End(r/id) as ring
↦ endid[1]=0
map psi = r,endphi;// defines the canonical map r/id -> End(r/id)
psi;
↦ psi[1]=T(1)^2
↦ psi[2]=T(1)^3
L[3];              // contribution to delta
↦ 1
```

D.4.23.7 genus

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Return: an integer, the geometric genus p_g = p_a - delta of the projective curve defined by i, where p_a is the arithmetic genus.

Note: genus always treats projective curves and takes projective closure if input is affine 1-dim variety. delta is the sum of all local delta-invariants of the singularities, i.e. dim(R'/R), R' the normalization of the local ring R of the singularity.
genus(I,"nor") uses the normalization to compute delta. Usually genus(I,"nor") is slower than genus(I) but sometimes not.
genus(I,"pri") starts with a primary decompsition.

Example:

```
LIB "normal.lib";
ring r=0,(x,y),dp;
ideal i=y^9 - x^2*(x - 1)^9;
genus(i);
↦ 0
ring r7=7,(x,y),dp;
ideal i=y^9 - x^2*(x - 1)^9;
genus(i);
↦ 0
```

D.4.23.8 primeClosure

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: primeClosure(L [,c]); L a list of a ring containing a prime ideal ker, c an optional integer

Return: a list L (of size n+1) consisting of rings L[1],...,L[n] such that - L[1] is a copy of (not a reference to!) the input ring L[1] - all rings L[i] contain ideals ker, L[2],...,L[n] contain ideals phi such that
L[1]/ker –> ... –> L[n]/ker

are injections given by the corresponding ideals phi, and L[n]/ker is the integral closure of L[1]/ker in its quotient field. - all rings L[i] contain a polynomial nzd such that elements of L[i]/ker are quotients of elements of L[i-1]/ker with denominator nzd via the injection phi.
L[n+1] is the delta invariant

Note: - L is constructed by recursive calls of primeClosure itself. - c determines the choice of nzd:
- c not given or equal to 0: first generator of the ideal SL, the singular locus of Spec(L[i]/ker)
- c<>0: the generator of SL with least number of monomials.

Example:

```
LIB "normal.lib";
ring R=0,(x,y),dp;
ideal I=x4,y4;
def K=ReesAlgebra(I)[1];        // K contains ker such that K/ker=R[It]
list L=primeClosure(K);
def R(1)=L[1];                  // L[4] contains ker, L[4]/ker is the
def R(4)=L[4];                  // integral closure of L[1]/ker
setring R(1);
R(1);
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    x y U(1) U(2)
↦ //        block   2 : ordering C
ker;
↦ ker[1]=y^4*U(1)-x^4*U(2)
setring R(4);
R(4);
↦ // coefficients: QQ
↦ // number of vars : 10
↦ //        block   1 : ordering a
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) T(2) T(\
   3) T(4)
↦ //                  : weights     1    1    1    1    3    2    2    1   \
    1    1
↦ //        block   2 : ordering dp
↦ //                  : names    X(1) X(2) X(3) X(4) X(5) X(6) X(7) T(2) T(\
   3) T(4)
↦ //        block   3 : ordering C
ker;
↦ ker[1]=X(2)*X(7)-X(5)
↦ ker[2]=X(1)*X(6)-X(5)
↦ ker[3]=X(1)^2*X(4)-X(2)*X(6)
↦ ker[4]=X(2)^2*X(3)-X(1)*X(7)
↦ ker[5]=X(1)*X(4)*X(7)-X(6)^2
↦ ker[6]=X(1)*X(2)*X(3)*X(4)-X(6)*X(7)
↦ ker[7]=X(2)*X(3)*X(6)-X(7)^2
↦ ker[8]=X(1)*T(3)-X(2)*T(4)
↦ ker[9]=-X(2)*T(2)+X(1)*T(4)
↦ ker[10]=-X(2)*T(2)+X(7)
↦ ker[11]=-X(2)*T(4)+X(6)
↦ ker[12]=X(6)*T(2)-X(7)*T(4)
↦ ker[13]=X(2)*X(4)*T(2)-X(6)*T(3)
↦ ker[14]=X(1)*X(4)-X(2)*T(3)
↦ ker[15]=X(1)*X(4)*T(2)-X(6)*T(4)
↦ ker[16]=X(2)*X(3)-X(1)*T(2)
↦ ker[17]=-X(2)^2*T(2)+X(5)
↦ ker[18]=X(3)*X(6)-X(7)*T(2)
↦ ker[19]=X(2)*X(3)*X(4)-X(7)*T(3)
↦ ker[20]=X(1)*X(3)*X(4)-X(7)*T(4)
↦ ker[21]=X(2)^2*X(6)*T(4)-X(2)*X(5)*T(3)
↦ ker[22]=T(2)^2-X(3)*T(4)
↦ ker[23]=-X(3)*X(4)+T(2)*T(3)
↦ ker[24]=T(3)^2-X(4)*T(4)
↦ ker[25]=-X(3)*T(3)+T(2)*T(4)
↦ ker[26]=-X(4)*T(2)+T(3)*T(4)
↦ ker[27]=-X(3)*X(4)+T(4)^2
```

D.4.23.9 closureFrac

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: closureFrac (L); L a list of size n+1 as in the result of primeClosure, L[n] contains an additional polynomial f

Create: a list fraction of two elements of L[1], such that
f=fraction[1]/fraction[2] via the injections phi L[i]–>L[i+1].

Example:

```
LIB "normal.lib";
ring R=0,(x,y),dp;
ideal ker=x2+y2;
export ker;
list L=primeClosure(R);          // We normalize R/ker
for (int i=1;i<=size(L);i++) { def R(i)=L[i]; }
setring R(2);
kill R;
phi;                             // The map R(1)-->R(2)
↦ phi[1]=X(1)
↦ phi[2]=X(2)
poly f=T(2);                     // We will get a representation of f
export f;
L[2]=R(2);
closureFrac(L);
setring R(1);
kill R(2);
fraction;                        // f=fraction[1]/fraction[2] via phi
↦ [1]:
↦    x
↦ [2]:
↦    y
kill R(1);
```

D.4.23.10 iMult

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: iMult(L); L a list of ideals

Return: int, the intersection multiplicity of the ideals of L; if iMult(L) is infinite, -1 is returned.

Theory: If r=size(L)=2 then iMult(L) = vdim(std(L[1]+L[2])) and in general iMult(L) = sum{ iMult(L[j],Lj) | j=1..r-1 } with Lj the intersection of L[j+1],...,L[r]. If I is the intersection of all ideals in L then we have delta(I) = delta(L[1])+...+delta(L[r]) + iMult(L) where delta(I) = vdim (normalisation(R/I)/(R/I)), R the basering.

Example:

```
LIB "normal.lib";
ring s  = 23,(x,y),dp;
list L = (x-y),(x3+y2);
iMult(L);
↦ 3
L = (x-y),(x3+y2),(x3-y4);
iMult(L);
↦ 19
```

D.4.23.11 deltaLoc

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: deltaLoc(f,J); f poly, J ideal

Assume: f is reduced bivariate polynomial; basering has exactly two variables; J is irreducible prime component of the singular locus of f (e.g., one entry of the output of `minAssGTZ(I);`, I = <f,jacob(f)>).

Return: list L:

`L[1]`; int: the sum of (local) delta invariants of f at the (conjugated) singular points given by J.

`L[2]`; int: the sum of (local) Tjurina numbers of f at the (conjugated) singular points given by J.

`L[3]`; int: the sum of (local) number of branches of f at the (conjugated) singular points given by J.

Note: procedure makes use of `execute`; increasing printlevel displays more comments (default: printlevel=0).

Example:

```
LIB "normal.lib";
ring r=0,(x,y),dp;
poly f=(x2+y^2-1)^3 +27x2y2;
ideal I=f,jacob(f);
I=std(I);
list qr=minAssGTZ(I);
size(qr);
↦ 6
// each component of the singular locus either describes a cusp or a pair
// of conjugated nodes:
```

```
deltaLoc(f,qr[1]);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    1
deltaLoc(f,qr[2]);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    1
deltaLoc(f,qr[3]);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    1
deltaLoc(f,qr[4]);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    1
deltaLoc(f,qr[5]);
↦ [1]:
↦    2
↦ [2]:
↦    2
↦ [3]:
↦    4
deltaLoc(f,qr[6]);
↦ [1]:
↦    2
↦ [2]:
↦    2
↦ [3]:
↦    4
```

See also: Section D.6.15.12 [delta], page 1730; Section D.6.20.15 [tjurina], page 1763.

D.4.23.12 locAtZero

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: locAtZero(I); I = ideal

Return: int, 1 if I has only one point which is located at zero, 0 otherwise

Assume: I is given as a standard bases in the basering

Note: only useful in affine rings, in local rings vdim does the check

Example:

```
LIB "normal.lib";
ring r = 0,(x,y,z),dp;
poly f = z5+y4+x3+xyz;
ideal i = jacob(f),f;
i=std(i);
locAtZero(i);
↦ 1
i= std(i*ideal(x-1,y,z));
locAtZero(i);
↦ 0
```

D.4.23.13 norTest

Procedure from library normal.lib (see Section D.4.23 [normal_lib], page 1183).

Assume: nor is the output of normal(i) (any options) or normalP(i,"withRing") or normalC(i) (any options). In particular, the ring nor[1][1] contains the ideal norid and the map normap: basering/i –> nor[1][1]/norid.

Return: an intvec v such that:

v[1] = 1 if the normap is injective and 0 otherwise
v[2] = 1 if the normap is finite and 0 otherwise
v[3] = 1 if nor[1][1]/norid is normal and 0 otherwise

If n=1 (resp n=2) only v[1] (resp. v[2]) is computed and returned

Theory: The procedure can be used to test whether the computation of the normalization was correct: basering/i –> nor[1][1]/norid is the normalization of basering/i if and only if v=1,1,0.

Note: For big examples it can be hard to fully test correctness; the partial test norTest(i,nor,2) is usually fast

Example:

```
LIB "normal.lib";
int prl = printlevel;
printlevel = -1;
ring r = 0,(x,y),dp;
ideal i = (x-y^2)^2 - y*x^3;
list nor = normal(i);
norTest(i,nor);                //1,1,1 means that normal was correct
↦ 1
↦ 1
↦ 1,1,1
nor = normalC(i);
norTest(i,nor);                //1,1,1 means that normal was correct
↦ 1
↦ 1
↦ 1,1,1
ring s = 2,(x,y),dp;
ideal i = (x-y^2)^2 - y*x^3;
nor = normalP(i,"withRing");
norTest(i,nor);               //1,1,1 means that normalP was correct
```

```
↦ 1
↦ 1
↦ 1,1,1
printlevel = prl;
```

D.4.23.14 getSmallest

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: getSmallest(J); J is an ideal.

Return: the generator of J of smallest degree. If there are more than one, it chooses the one with smallest number of monomials.

Note: It looks only at the generator of J, not at all the polynomials in the ideal.
It is intended maninly to compute a good universal denominator in the normalization algorithms.

Example:

```
LIB "normal.lib";
printlevel = printlevel+1;
ring s = 0,(x,y),dp;
ideal J = x3-y, y5, x2-y2+1;
getSmallest(J);
↦ x2-y2+1
printlevel = printlevel-1;
```

D.4.23.15 getOneVar

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: getOneVar(J, vari); J is a 0-dimensional ideal, vari is an integer.

Return: a polynomial of J in the variable indicated by vari of smallest degree.

Note: Works only over rings of two variables.
It is intended mainly as an auxiliary procedure for computing integral bases.

Example:

```
LIB "normal.lib";
printlevel = printlevel+1;
ring s = 0,(x,y),dp;
ideal J = x3-y, y3;
getOneVar(J, 1);
↦ x9
printlevel = printlevel-1;
```

D.4.23.16 changeDenominator

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: changeDenominator(U1, c1, c2, I); U1 and I ideals, c1 and c2 polynomials.

Return: an ideal U2 such that the A-modules 1/c1 * U1 and 1/c2 * U2 are equal, where A = R/I and R is the basering.

Note: It assumes that such U2 exists. It is intended maninly as an auxiliary procedure in the normalization algorithms.

Example:

```
LIB "normal.lib";
ring s = 0,(x,y),dp;
ideal I = y5-y4x+4y2x2-x4;
ideal U1 = normal(I)[2][1];
↦
↦ // 'normal' created a list, say nor, of two elements.
↦ // To see the list type
↦       nor;
↦
↦ // * nor[1] is a list of 1 ring(s).
↦ // To access the i-th ring nor[1][i], give it a name, say Ri, and type
↦      def R1 = nor[1][1]; setring R1; norid; normap;
↦ // For the other rings type first (if R is the name of your base ring)
↦      setring R;
↦ // and then continue as for R1.
↦ // Ri/norid is the affine algebra of the normalization of R/P_i where
↦ // P_i is the i-th component of a decomposition of the input ideal id
↦ // and normap the normalization map from R to Ri/norid.
↦
↦ // * nor[2] is a list of 1 ideal(s). Let ci be the last generator
↦ // of the ideal nor[2][i]. Then the integral closure of R/P_i is
↦ // generated as R-submodule of the total ring of fractions by
↦ // 1/ci * nor[2][i].
poly c1 = U1[4];
U1;c1;
↦ U1[1]=xy2
↦ U1[2]=x2y
↦ U1[3]=x3
↦ U1[4]=y3
↦ y3
// 1/c1 * U1 is the normalization of I.
ideal U2 = changeDenominator(U1, c1, x3, I);
U2;
↦ U2[1]=-xy3+y4+4x2y
↦ U2[2]=-x2y2+xy3+4x3
↦ U2[3]=-x3y+x2y2-4xy3+4y4+16x2y
↦ U2[4]=x3
// 1/x3 * U2 is also the normalization of I, but with a different denominator.
```

D.4.23.17 normalConductor

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: normalConductor(I); I ideal

Assume: I is a radical ideal

Return: the conductor of R/I as ideal in R

Remarks: The procedures makes use of the minimal primes and
the generators of the normalization given by the normalization algorithm.

Note: the optional parameter can be used if the normalization has already been computed. If a list L contains the output of the procedure normal (with options prim, wd and usering if the ring has a mixed ordering), apply normalConductor(I,L)

Example:

```
LIB "normal.lib";
/////////////////////////////////////////////
// Computation of small conductor ideals //
/////////////////////////////////////////////
ring R = 0,(x,y,z),ds;
ideal I = x2y2 - z;
normalConductor(I);
↦ _[1]=1
// The conductor is the whole ring - so the ring is normal
// We can also see this using the delta invariant:
curveDeltaInv(I);
↦ 0
ring S = 0,(a,b,c),dp;
ideal J = abc;
normalConductor(J);
↦ _[1]=bc
↦ _[2]=ac
↦ _[3]=ab
// The conductor is not the whole ring - so it is not normal
// We can also see this using the delta invariant, which is even infinite
curveDeltaInv(J);
↦ -1
kill R,S;
/////////////////////////////////////////
// Computation of a bigger example //
/////////////////////////////////////////
ring R = 0,(x,y,z,t),ds;
ideal I = xyz - yzt, x2y3 - z2t4;
I = std(radical(I));
// Ideal I
I;
↦ I[1]=x2y-xyt
↦ I[2]=xyz-yzt
↦ I[3]=xzt-zt2
↦ I[4]=xy3-z2t3
↦ I[5]=y3zt-z3t3
// Conductor
normalConductor(I);
↦ _[1]=xy
↦ _[2]=xz-zt
↦ _[3]=yt
↦ _[4]=x2t-xt2
↦ _[5]=zt2
↦ _[6]=y3z
```

See also: Section D.4.5.2 [curveConductorMult], page 1049.

D.4.23.18 isNormal

Procedure from library `normal.lib` (see Section D.4.23 [normal_lib], page 1183).

Usage: isNormal(I); I ideal.

Return: 1 if R/I is normal and 0 if R/I is not normal, where R is the basering.

Example:

```
LIB "normal.lib";
ring R = 0, (x,y), dp;
ideal I1 = x2 - y3;
isNormal(I1);
↦ 0
ideal I2 = x - y3;
isNormal(I2);
↦ 1
```

D.4.24 normaliz_lib

Library: normaliz.lib

Purpose: Provides an interface for the use of Normaliz 3.10.0 or newer within SINGULAR.

Authors: Winfried Bruns, wbruns@uos.de
Christof Soeger, Christof.Soeger@Uni-Osnabrueck.de

Overview: The library normaliz.lib provides an interface for the use of Normaliz 3.10.0 or newer within SINGULAR. The exchange of data is via files. In addition to the top level functions that aim at objects of type ideal or ring, several other auxiliary functions allow the user to apply Normaliz to data of type intmat. Options such as compoutationn goals or algorithmic variants can be activated.To some extent, SINGULAR can therefore be used as an environment for interactive access to Normaliz.
Please see the `Normaliz.pdf` (included in the Normaliz distribution) for a more extensive documentation of Normaliz.

Normaliz allows the use of a grading. In the Singular functions that access Normaliz the parameter grading is an intvec that assigns a (not necessarily positive) degree to every variable of the ambient polynomial ring. But it must give positive degrees to the generators given to the function.

Singular and Normaliz exchange data via files. The input files use the Normaliz 3 syntax. These files are automatically created and erased behind the scenes. As long as one wants to use only the ring-theoretic functions there is no need for file management.
Note that the numerical invariants computed by Normaliz can be accessed in this "automatic file mode".
However, if Singular is used as a frontend for Normaliz or the user wants to inspect data not automatically returned to Singular, then an explicit filename and a path can be specified for the exchange of data. Moreover, the library provides functions for access to these files. Deletion of the files is left to the user. (Not all output files of Normaliz can be read by this library.)

Use of this library requires the program Normaliz to be installed. You can download it from `https://github.com/Normaliz/Normaliz/releases`. Please make sure that the executable is in the search path or use setNmzExecPath (Section D.4.24.23 [setNmzExecPath], page 1227).

Procedures:

D.4.24.1 intclToricRing

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intclToricRing(ideal I);
intclToricRing(ideal I, intvec grading);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I (considered as a list of polynomials). The function computes the integral
closure T of S in the basering and returns an ideal listing the algebra generators of T over the coefficient field.
The function returns the input ideal I if an option blocking the computation of Hilbert bases has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Note: A mathematical remark: the toric ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
LIB "normaliz.lib";
ring R=37,(x,y,t),dp;
ideal I=x3,x2y,y3;
intclToricRing(I);
↦ _[1]=y
↦ _[2]=x
showNuminvs();
↦ hilbert_basis_elements : 2
↦ number_extreme_rays : 2
↦ dim_max_subspace : 0
↦ embedding_dim : 3
↦ rank : 2
↦ external_index : 1
↦ internal_index : 3
↦ number_support_hyperplanes : 2
↦ size_triangulation : 1
↦ sum_dets : 1
↦ integrally_closed : 0
↦ inhomogeneous : 0
↦ graded : 1
↦ degree_1_elements : 2
↦ grading : 1,1,0
↦ grading_denom : 1
↦ multiplicity : 1
↦ multiplicity_denom : 1
↦ hilbert_series_num : 1
↦ hilbert_series_denom : 1,1
↦ class_group : 0
//now the same example with another grading
intvec grading = 2,3,1;
```

```
intclToricRing(I,grading);
↦ _[1]=x
↦ _[2]=y
showNuminvs();
↦ hilbert_basis_elements : 2
↦ number_extreme_rays : 2
↦ dim_max_subspace : 0
↦ embedding_dim : 3
↦ rank : 2
↦ external_index : 1
↦ internal_index : 3
↦ number_support_hyperplanes : 2
↦ size_triangulation : 1
↦ sum_dets : 1
↦ integrally_closed : 0
↦ inhomogeneous : 0
↦ graded : 1
↦ degree_1_elements : 0
↦ grading : 2,3,1
↦ grading_denom : 1
↦ multiplicity : 1
↦ multiplicity_denom : 6
↦ hilbert_series_num : 1,-1,1
↦ hilbert_series_denom : 1,6
↦ class_group : 0
```

See also: Section D.4.24.5 [ehrhartRing], page 1213; Section D.4.24.6 [intclMonIdeal], page 1214; Section D.4.24.2 [normalToricRing], page 1211.

D.4.24.2 normalToricRing

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: normalToricRing(ideal I);
normalToricRing(ideal I, intvec grading);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I (considered as a list of polynomials). The function computes the normalisation T of S and returns an ideal listing the algebra generators of T over the coefficient field.
The function returns the input ideal I if one of the options blocking the computation of Hilbert bases has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Note: A mathematical remark: the toric ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
LIB "normaliz.lib";
ring  R = 37,(x,y,t),dp;
ideal I = x3,x2y,y3;
normalToricRing(I);
↦ _[1]=y3
↦ _[2]=xy2
↦ _[3]=x2y
↦ _[4]=x3
```

See also: Section D.4.24.5 [ehrhartRing], page 1213; Section D.4.24.6 [intclMonIdeal], page 1214; Section D.4.24.1 [intclToricRing], page 1210; Section D.4.24.3 [normalToricRingFromBinomials], page 1212; Section D.4.24.4 [toricRingFromBinomials], page 1213.

D.4.24.3 normalToricRingFromBinomials

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: normalToricRingFromBinomials(ideal I);
normalToricRingFromBinomials(ideal I, intvec grading);

Return: The ideal I is generated by binomials of type $X^a - X^b$ (multiindex notation) in the surrounding polynomial ring $K[X] = K[X_1, ..., X_n]$. The binomials represent a congruence on the monoid Z^n with residue monoid M. Let N be the image of M in gp(M)/torsion. Then N is universal in the sense that every homomorphism from M to an affine monoid factors through N. If I is a prime ideal, then $K[N] = K[X]/I$. In general, $K[N] = K[X]/P$ where P is the unique minimal prime ideal of I generated by binomials of type $X^a - X^b$.

The function computes the normalization of $K[N]$ and returns a newly created polynomial ring of the same Krull dimension, whose variables are $x(1), ..., x(n-r)$, where r is the rank of the matrix with rows $a-b$. (In general there is no canonical choice for such an embedding.) Inside this polynomial ring there is an ideal I which lists the algebra generators of the normalization of $K[N]$.

The function returns the input ideal I if an option blocking the computation of Hilbert bases has been activated.
However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Example:

```
LIB "normaliz.lib";
ring R = 37,(u,v,w,x,y,z),dp;
ideal I = u2v-xyz, ux2-wyz, uvw-y2z;
def S = normalToricRingFromBinomials(I);
setring S;
I;
↦ I[1]=x(3)
↦ I[2]=x(1)
↦ I[3]=x(2)*x(3)^3
↦ I[4]=x(1)*x(2)*x(3)^2
↦ I[5]=x(1)^2*x(2)*x(3)
↦ I[6]=x(1)^3*x(2)
↦ I[7]=x(1)*x(2)^2*x(3)^4
↦ I[8]=x(1)^2*x(2)^2*x(3)^3
↦ I[9]=x(1)^2*x(2)^3*x(3)^5
```

See also: Section D.4.24.5 [ehrhartRing], page 1213; Section D.4.24.6 [intclMonIdeal], page 1214; Section D.4.24.1 [intclToricRing], page 1210; Section D.4.24.2 [normalToricRing], page 1211.

D.4.24.4 toricRingFromBinomials

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: toricRingFromBinomials(ideal I);
toricRingFromBinomials(ideal I, intvec grading);

Return: The ideal I is generated by binomials of type $X^a - X^b$ (multiindex notation) in the surrounding polynomial ring $K[X] = K[X_1, ..., X_n]$. The binomials represent a congruence on the monoid Z^n with residue monoid M. Let N be the image of M in gp(M)/torsion. Then N is universal in the sense that every homomorphism from M to an affine monoid factors through N. If I is a prime ideal, then $K[N] = K[X]/I$. In general, $K[N] = K[X]/P$ where P is the unique minimal prime ideal of I generated by binomials of type $X^a - X^b$.

The function computes $K[N]$ and returns a newly created polynomial ring of the same Krull dimension, whose variables are $x(1), ..., x(n-r)$, where r is the rank of the matrix with rows $a - b$. (In general there is no canonical choice for such an embedding.)

The function returns the input ideal I if an option blocking the computation of Hilbert bases has been activated.
However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Example:

```
LIB "normaliz.lib";
ring R = 37,(u,v,w,x,y,z),dp;
ideal I = u2v-xyz, ux2-wyz, uvw-y2z;
def S = toricRingFromBinomials(I);
setring S;
I;
↦ I[1]=x(3)
↦ I[2]=x(1)
↦ I[3]=x(2)*x(3)^3
↦ I[4]=x(1)*x(2)*x(3)^2
↦ I[5]=x(1)^3*x(2)
↦ I[6]=x(1)^2*x(2)^3*x(3)^5
```

See also: Section D.4.24.3 [normalToricRingFromBinomials], page 1212; Section D.4.24.4 [toricRingFromBinomials], page 1213.

D.4.24.5 ehrhartRing

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: ehrhartRing(ideal I);

Return: The exponent vectors of the leading monomials of the elements of I are considered as points of a lattice polytope P.
The Ehrhart ring of a (lattice) polytope P is the monoid algebra defined by the monoid of lattice points in the cone over the polytope P; see Bruns and Gubeladze, Polytopes, Rings, and K-theory, Springer 2009, pp. 228, 229.
The function returns a list of ideals:
(i) If the last ring variable is not used by the monomials, it is treated as the auxiliary

variable of the Ehrhart ring. The function returns two ideals, the first containing the monomials representing all the lattice points of the polytope, the second containing the algebra generators of the Ehrhart ring over the coefficient field.
(ii) If the last ring variable is used by the monomials, the list returned contains only one ideal, namely the monomials representing the lattice points of the polytope.

The function returns the a list containing the input ideal I if an option blocking the computation of the Hilbert basis has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Note: A mathematical remark: the Ehrhart ring depends on the list of monomials given, and not only on the ideal they generate!

Example:

```
LIB "normaliz.lib";
ring R=37,(x,y,t),dp;
ideal J=x3,x2y,y3,xy2t7;
ehrhartRing(J);
↦ [1]:
↦    _[1]=y3
↦    _[2]=xy2
↦    _[3]=xy2t
↦    _[4]=xy2t2
↦    _[5]=xy2t3
↦    _[6]=xy2t4
↦    _[7]=xy2t5
↦    _[8]=xy2t6
↦    _[9]=xy2t7
↦    _[10]=x2y
↦    _[11]=x2yt
↦    _[12]=x2yt2
↦    _[13]=x2yt3
↦    _[14]=x3
```

See also: Section D.4.24.6 [intclMonIdeal], page 1214; Section D.4.24.1 [intclToricRing], page 1210; Section D.4.24.2 [normalToricRing], page 1211.

D.4.24.6 intclMonIdeal

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intclMonIdeal(ideal I);
intclMonIdeal(ideal I, intvec grading);

Return: The exponent vectors of the leading monomials of the elements of I are considered as generators of a monomial ideal for which the normalization of its Rees algebra is computed. For a Definition of the Rees algebra (or Rees ring) see Bruns and Herzog, Cohen-Macaulay rings, Cambridge University Press 1998, p. 182.
The function returns a list of ideals:
(i) If the last ring variable is not used by the monomials, it is treated as the auxiliary variable of the Rees algebra. The function returns two ideals, the first containing the monomials generating the integral closure of the monomial ideal, the second containing

the algebra generators of the normalization of the Rees algebra.
(ii) If the last ring variable is used by the monomials, the list returned contains only one ideal, namely the monomials generating the integral closure of the ideal.
The function returns the a list containing the input ideal I if an option blocking the computation of Hilbert bases has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,t),dp;
ideal I=x^2,y^2,z^3;
list l=intclMonIdeal(I);
l[1]; // integral closure of I
↦ _[1]=z3
↦ _[2]=yz2
↦ _[3]=y2
↦ _[4]=xz2
↦ _[5]=xy
↦ _[6]=x2
l[2];  // monomials generating the integral closure of the Rees algebra
↦ _[1]=z
↦ _[2]=z3t
↦ _[3]=y
↦ _[4]=yz2t
↦ _[5]=y2t
↦ _[6]=x
↦ _[7]=xz2t
↦ _[8]=xyt
↦ _[9]=x2t
```

See also: Section D.4.24.5 [ehrhartRing], page 1213; Section D.4.24.1 [intclToricRing], page 1210; Section D.4.24.2 [normalToricRing], page 1211.

D.4.24.7 definingBinomialIdeal

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: definingBinomialIdeal(ideal I);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I (considered as a list of polynomials). The function computes the computes the definig binomial ideal J of S with respect to the generators and returns it, together with J. DSee eample.

Note: A mathematical remark: the toric ring depends on the list of monomials given, and not only on the ideal they generate! This function requires the previous setting of an NmzFilename. The function does not delete the written files.

Example:

```
LIB "normaliz.lib";
ring R = 37,(x,y,z,w),dp;
ideal I = x4,x3y,x2y2,xy3,y4;
setNmzFilename("binomials");
```

```
def S = definingBinomialIdeal(I);
↦ 1
↦ 1
setring S;
J;
↦ J[1]=-x(3)*x(4)+x(2)*x(5)
↦ J[2]=-x(2)*x(4)+x(1)*x(5)
↦ J[3]=x(2)*x(3)-x(1)*x(4)
↦ J[4]=-x(2)^2+x(1)*x(3)
↦ J[5]=-x(4)^2+x(3)*x(5)
↦ J[6]=x(3)^2-x(2)*x(4)
```

See also: Section D.4.24.3 [normalToricRingFromBinomials], page 1212.

D.4.24.8 latticeIdeal

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: latticeIdeal(ideal I);

Return: Returns the lattice ideal defined by the elements of I which have to be binomials. The lattice ideal is
the restriction to the polynomial ring of the ideal of the Laurent polynomial ring generated by I

Note: This function requires the previous setting of an NmzFilename. The function does not delete the written files.

Example:

```
LIB "normaliz.lib";
setNmzFilename("binomials");
ring S = 37,(u,v,w,x,y,z),dp;
ideal I = u2-v2, x2-y2, y2-vw, z2-xy;
latticeIdeal(I);
↦ 1
↦ 1
↦ _[1]=x2-y2
↦ _[2]=u2-v2
↦ _[3]=-xy+z2
↦ _[4]=vw-y2
```

See also: Section D.4.24.34 [intmat2binomials], page 1232.

D.4.24.9 groebnerBasis

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: groebnerBasis(ideal I);

Return: Returns a Gr"obner basis of the lattice ideal defined by the elements of I which have to be binomials. The lattice ideal is the restriction to the polynomial ring of the ideal of the Laurent polynomial ring generated by I. The default monomial order is DegRevLex where Deg is the total degree on the ambient polynomial ring. Lex and DegLex orders can be chosen via options.

Note: This function requires the previous setting of an NmzFilename. The function does not delete the written files.

Example:

```
LIB "normaliz.lib";
setNmzFilename("binomials");
ring S = 37,(u,v,w,x,y,z),dp;
ideal I = u2-v2, x2-y2, y2-vw, z2-xy;
groebnerBasis(I);
↦ _[1]=xy-z2
↦ _[2]=x2-y2
↦ _[3]=vw-y2
↦ _[4]=u2-v2
↦ _[5]=y3-xz2
setNmzOption("lex",1);
↦ 1
groebnerBasis(I);
↦ _[1]=y4-z4
↦ _[2]=-y3+xz2
↦ _[3]=xy-z2
↦ _[4]=x2-y2
↦ _[5]=vw-y2
↦ _[6]=u2-v2
setNmzOption("lex",0);
↦ 1
```

See also: Section D.4.24.8 [latticeIdeal], page 1216.

D.4.24.10 torusInvariants

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: torusInvariants(intmat A);
torusInvariants(intmat A, intvec grading);

Return: Returns an ideal representing the list of monomials generating the ring of invariants as an algebra over the coefficient field. R^T.
The function returns the ideal given by the input matrix A if one of the options `supp`, `triang`, `volume`, or `hseries` has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Background:
Let $T = (K^*)^r$ be the r-dimensional torus acting on the polynomial ring $R = K[X_1, \ldots, X_n]$ diagonally. Such an action can be described as follows: there are integers $a_{i,j}$, $i = 1, \ldots, r$, $j = 1, \ldots, n$, such that $(\lambda_1, \ldots, \lambda_r) \in T$ acts by the substitution

$$X_j \mapsto \lambda_1^{a_{1,j}} \cdots \lambda_r^{a_{r,j}} X_j, \quad j = 1, \ldots, n.$$

In order to compute the ring of invariants R^T one must specify the matrix $A = (a_{i,j})$.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,w),dp;
intmat E[2][4] = -1,-1,2,0, 1,1,-2,-1;
torusInvariants(E);
↦ _[1]=y2z
↦ _[2]=xyz
↦ _[3]=x2z
```

See also: Section D.4.24.12 [diagInvariants], page 1219; Section D.4.24.11 [finiteDiagInvariants], page 1218; Section D.4.24.14 [intersectionValRingIdeals], page 1220; Section D.4.24.13 [intersectionValRings], page 1220.

D.4.24.11 finiteDiagInvariants

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: finiteDiagInvariants(intmat U);
finiteDiagInvariants(intmat U, intvec grading);

Return: This function computes the ring of invariants of a finite abelian group G acting diagonally on the surrounding polynomial ring $K[X_1, ..., X_n]$. The group is the direct product of cyclic groups generated by finitely many elements $g_1, ..., g_w$. The element g_i acts on the indeterminate X_j by $g_i(X_j) = \lambda_i^{u_{ij}} X_j$ where λ_i is a primitive root of unity of order equal to $ord(g_i)$. The ring of invariants is generated by all monomials satisfying the system $u_{i1}a_1 + \ldots + u_{in}a_n \equiv 0$ mod $\mathrm{ord}(g_i)$, $i = 1, \ldots, w$. The input to the function is the $w \times (n+1)$ matrix U with rows $u_{i1} \ldots u_{in}$ $\mathrm{ord}(gi)$, $i = 1, \ldots, w$. The output is a monomial ideal listing the algebra generators of the subalgebra of invariants $R^G = \{f \in R : g_i f = f$ for all $i = 1, \ldots, w\}$.

The function returns the ideal given by the input matrix C if one of the options `supp`, `triang`, `volume`, or `hseries` has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Note:

Example:

```
LIB "normaliz.lib";
ring R = 0,(x,y,z,w),dp;
intmat C[2][5] = 1,1,1,1,5, 1,0,2,0,7;
finiteDiagInvariants(C);
↦ _[1]=w5
↦ _[2]=z7w3
↦ _[3]=z14w
↦ _[4]=z35
↦ _[5]=yw4
↦ _[6]=yz7w2
↦ _[7]=yz14
↦ _[8]=y2w3
↦ _[9]=y2z7w
↦ _[10]=y3w2
↦ _[11]=y3z7
↦ _[12]=y4w
↦ _[13]=y5
↦ _[14]=xz3w
↦ _[15]=xz24
↦ _[16]=xyz3
↦ _[17]=x2z13
↦ _[18]=x3z2
↦ _[19]=x5zw4
↦ _[20]=x5yzw3
↦ _[21]=x5y2zw2
↦ _[22]=x5y3zw
↦ _[23]=x5y4z
↦ _[24]=x7w3
↦ _[25]=x7yw2
↦ _[26]=x7y2w
↦ _[27]=x7y3
↦ _[28]=x12zw2
↦ _[29]=x12yzw
↦ _[30]=x12y2z
↦ _[31]=x14w
↦ _[32]=x14y
↦ _[33]=x19z
↦ _[34]=x35
```

See also: Section D.4.24.12 [diagInvariants], page 1219; Section D.4.24.14 [intersectionValRingIdeals], page 1220; Section D.4.24.13 [intersectionValRings], page 1220; Section D.4.24.10 [torusInvariants], page 1217.

D.4.24.12 diagInvariants

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: diagInvariants(intmat A, intmat U);
diagInvariants(intmat A, intmat U, intvec grading);

Return: This function computes the ring of invariants of a diagonalizable group $D = T \times G$ where T is a torus and G is a finite abelian group, both acting diagonally on the polynomial ring $K[X_1, \ldots, X_n]$. The group actions are specified by the input matrices A and U. The first matrix specifies the torus action, the second the action of the finite group. See torusInvariants and finiteDiagInvariants for more detail. The output is a monomial ideal listing the algebra generators of the subalgebra of invariants.

The function returns the ideal given by the input matrix A if one of the options `supp`, `triang`, `volume`, or `hseries` has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,w),dp;
intmat E[2][4] = -1,-1,2,0, 1,1,-2,-1;
intmat C[2][5] = 1,1,1,1,5, 1,0,2,0,7;
diagInvariants(E,C);
↦ _[1]=y70z35
↦ _[2]=xy19z10
↦ _[3]=x4y6z5
↦ _[4]=x15y5z10
↦ _[5]=x26y4z15
↦ _[6]=x37y3z20
↦ _[7]=x48y2z25
↦ _[8]=x59yz30
↦ _[9]=x70z35
```

See also: Section D.4.24.11 [finiteDiagInvariants], page 1218; Section D.4.24.14 [intersectionValRingIdeals], page 1220; Section D.4.24.13 [intersectionValRings], page 1220; Section D.4.24.10 [torusInvariants], page 1217.

D.4.24.13 intersectionValRings

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intersectionValRings(intmat V, intvec grading);

Return: The function returns a monomial ideal, to be considered as the list of monomials generating S as an algebra over the coefficient field.

Background:

A discrete monomial valuation v on $R = K[X_1, \ldots, X_n]$ is determined by the values $v(X_j)$ of the indeterminates. This function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0,\ i = 1, \ldots, r\}$ for several such valuations v_i, $i = 1, \ldots, r$. It needs the matrix $V = (v_i(X_j))$ as its input.

The function returns the ideal given by the input matrix V if one of the options `supp`, `triang`, `volume`, or `hseries` has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,w),dp;
intmat V0[2][4]=0,1,2,3, -1,1,2,1;
intersectionValRings(V0);
↦ _[1]=w
↦ _[2]=z
↦ _[3]=y
↦ _[4]=xw
↦ _[5]=xz
↦ _[6]=xy
↦ _[7]=x2z
```

See also: Section D.4.24.12 [diagInvariants], page 1219; Section D.4.24.11 [finiteDiagInvariants], page 1218; Section D.4.24.14 [intersectionValRingIdeals], page 1220; Section D.4.24.10 [torusInvariants], page 1217.

D.4.24.14 intersectionValRingIdeals

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intersectionValRingIdeals(intmat V);
intersectionValRingIdeals(intmat V, intvec grading);

Return: The function returns two ideals, both to be considered as lists of monomials. The first is the system of monomial generators of S , the second the system of generators of M
.

The function returns a list consisting of the ideal given by the blocking the computation of Hilbert bases has been activated. However, in this case some numerical invariants are computed, and some other data may be contained in files that you can read into Singular (see Section D.4.24.15 [showNuminvs], page 1221, Section D.4.24.16 [exportNuminvs], page 1222).

Background:

A discrete monomial valuation v on $R = K[X_1, \ldots, X_n]$ is determined by the values $v(X_j)$ of the indeterminates. This function computes the subalgebra $S = \{f \in R : v_i(f) \geq 0,\ i = 1, \ldots, r\}$ for several such valuations v_i, $i = 1, \ldots, r$. It needs the matrix $V = (v_i(X_j))$ as its input.

This function simultaneously determines the S-submodule $M = \{f \in R : v_i(f) \geq w_i,\ i = 1, \ldots, r\}$ for integers $w_1, \ldots\ w_r$. (If $w_i \geq 0$ for all i, M is an ideal of S.) The numbers w_i form the $(n + 1)$th column of the input matrix.

Note: The function also gives an error message if the matrix V has the wrong number of columns.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,w),dp;
intmat V[2][5]=0,1,2,3,4, -1,1,2,1,3;
intersectionValRingIdeals(V);
↦ [1]:
↦    _[1]=w
↦    _[2]=z
↦    _[3]=y
↦    _[4]=xw
↦    _[5]=xz
↦    _[6]=xy
↦    _[7]=x2z
↦ [2]:
↦    _[1]=w3
↦    _[2]=zw
↦    _[3]=z2
↦    _[4]=yw2
↦    _[5]=y2w
↦    _[6]=y2z
↦    _[7]=y4
↦    _[8]=xz2
↦    _[9]=xy2z
↦    _[10]=xy4
```

See also: Section D.4.24.12 [diagInvariants], page 1219; Section D.4.24.11 [finiteDiagInvariants], page 1218; Section D.4.24.13 [intersectionValRings], page 1220; Section D.4.24.10 [torusInvariants], page 1217.

D.4.24.15 showNuminvs

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: showNuminvs();

Purpose: prints the numerical invariants

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,t),dp;
ideal I=x3,x2y,y3;
list l=intclMonIdeal(I);
showNuminvs();
↦ hilbert_basis_elements : 7
↦ number_extreme_rays : 5
↦ dim_max_subspace : 0
↦ embedding_dim : 4
↦ rank : 4
↦ external_index : 1
↦ internal_index : 1
↦ number_support_hyperplanes : 5
↦ size_triangulation : 3
↦ sum_dets : 4
↦ integrally_closed : 0
↦ inhomogeneous : 0
↦ graded : 1
↦ degree_1_elements : 7
↦ grading : 1,1,1,-2
↦ grading_denom : 1
↦ multiplicity : 4
↦ multiplicity_denom : 1
↦ hilbert_series_num : 1,3
↦ hilbert_series_denom : 1,1,1,1
↦ primary : 0
↦ class_group : 1
```

See also: Section D.4.24.16 [exportNuminvs], page 1222.

D.4.24.16 exportNuminvs

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: exportNuminvs();

Create: Creates top-level variables which contain the numerical invariants. Depending on the options of normaliz different invariants are calculated. Use showNuminvs (Section D.4.24.15 [showNuminvs], page 1221) to see which invariants are available.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,t),dp;
ideal I=x3,x2y,y3;
list l=intclMonIdeal(I);
exportNuminvs();
// for example, now the following variables are set:
nmz_hilbert_basis_elements;
↦ 7
nmz_number_extreme_rays;
↦ 5
nmz_rank;
↦ 4
nmz_number_support_hyperplanes;
↦ 5
```

```
nmz_multiplicity;
↦ 4
nmz_primary;
↦ 0
```

See also: Section D.4.24.15 [showNuminvs], page 1221.

D.4.24.17 allNmzOptions

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: proc alNmzOptions();

Purpose: The function prints a list of the Normaliz options that are available in this library: the string naming the option, the default value, and the option sent to Normaliz.

Example:

```
LIB "normaliz.lib";
allNmzOptions();
↦ supp 0 -s
↦ triang 0 -T
↦ volume 0 -v
↦ hvect 0 -p  -- name deprecated, use hvect_deg1
↦ hvect_deg1 0 -p
↦ only_hvect 0 -q
↦ fvect 0 --FVector
↦ height1 0 -1
↦ Gorenst 0 -G
↦ intclosed 0 --IsIntegrallyClosed
↦ witness 0 --w
↦ classgroup 0 -C
↦ normal 0 -n  -- name deprecated, use hilbbasvol
↦ hilbbasvol 0 -n
↦ normal_l 0 -N  -- name deprecated, use hilbbas
↦ hilbbas 0 -N
↦ hilb 0 -h  -- name deprecated, use hilbbas_hvect
↦ hilbbas_hvect 0 -h
↦ dual 0 -d
↦ markov 0 --MarkovBasis --BinomialsPacked
↦ groebner 0 --GroebnerBasis --BinomialsPacked
↦ lex 0 --Lex
↦ revlex 0 --RevLex
↦ lex 0 --Lex
↦ deglex 0 --DegLex
↦ genoverori 0 -M --mod
↦ type 0 --typ
↦ control 0 -c  -- name deprecated, use verbose
↦ verbose 0 -c
↦ allf 0 -a
↦ bigint 0 -B
↦ threads 8 -x=
↦ errorcheck 0 -e  -- allowed, but ignored by Normaliz
↦
```

See also: Section D.4.24.18 [setNmzOption], page 1224; Section D.4.24.20 [showNmzOptions], page 1225.

D.4.24.18 setNmzOption

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: setNmzOption(string s, int onoff);

Purpose: If `onoff=1` the option `s` is activated, and if `onoff=0` it is deactivated. The predefined Normaliz options are accessible via the following names:
`-s: supp`
`-t: triang`
`-v: volume`
`-p: hvect` deprecated, replacement:
`-p: hvect_deg1`
`-q: only_hvect`
`--FVector: fvect`
`-1: height1`
`-G: Gorenst`
`-w: witness`
`--IsIntegrallyClosed: intclosed`
`-C: classgroup`
`-n: normal` deprecated, replacement:
`-n: hilbbasvol`
`-N: normal_l` deprecated, replacement:
`-N: hilbbas`
`-h: hilb` deprecated, replacement:
`-h: hilbbas_hvect`
`--MarkovBasis: groebner`
`--GroebnerBasis: markov`
`--Lex: lex`
`--RevLex: revlex`
`--DegLex: deglex`
`-d: dual`
`-M: genoverori`
`-a: allf`
`--typ: type`
`-c: control` deprecated, replacement:
`-c: verbose`
`-e: errorcheck` allowed, but ignored
`-B: bigint` Use GMP for arbitrary precision integers
`-x=N: threads` In this case the int parameter is used to set the number of threads N, deafult 8, 0 means no explicit limiting.

Further Normaliz otions can be adeded to tzhe nlist by addNmzOption.

Example:

```
LIB "normaliz.lib";
setNmzOption("only_hvect",1);
↦ 1
showNmzOptions();
↦  -f -q -x=8
setNmzOption("only_hvect",0);
↦ 1
showNmzOptions();
```

```
     ↦  -f -x=8
```

See also: Section D.4.24.19 [addNmzOption], page 1225; Section D.4.24.17 [allNmzOptions], page 1223; Section D.4.24.21 [resetNmzOptions], page 1225; Section D.4.24.20 [showNmzOptions], page 1225.

D.4.24.19 addNmzOption

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: addNmzOption(string short_cut, string nmz_option)

Note: This function allows the addition of Normaliz options
in addition to the predefined ones. Adding an option does not activate it.
Note: The function prefixes a single letter option by - and multiletter options by –.

Example:

```
     LIB "normaliz.lib";
     addNmzOption("FL", "FaceLattice");
     setNmzOption("FL", 1);
     ↦ 1
     showNmzOptions();
     ↦  -f -x=8 --FaceLattice
     resetNmzOptions();
```

See also: Section D.4.24.17 [allNmzOptions], page 1223; Section D.4.24.21 [resetNmzOptions], page 1225; Section D.4.24.18 [setNmzOption], page 1224; Section D.4.24.20 [showNmzOptions], page 1225.

D.4.24.20 showNmzOptions

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: showNmzOptions();

Return: Returns the string of activated options.

Note: This string is used as parameter when calling Normaliz.

Example:

```
     LIB "normaliz.lib";
     setNmzOption("hilb",1);
     ↦ 1
     showNmzOptions();
     ↦  -f -h -x=8
     setNmzOption("hilb",0);
     ↦ 1
     showNmzOptions();
     ↦  -f -x=8
```

See also: Section D.4.24.18 [setNmzOption], page 1224.

D.4.24.21 resetNmzOptions

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: resetNmzOptions();

Purpose: Resets the options to the dafault value.

Example:

```
LIB "normaliz.lib";
setNmzOption("only_hvect",1);
↦ 1
showNmzOptions();
↦  -f -q -x=8
resetNmzOptions();
showNmzOptions();
↦  -f -x=8
```

See also: Section D.4.24.19 [addNmzOption], page 1225; Section D.4.24.17 [allNmzOptions], page 1223; Section D.4.24.18 [setNmzOption], page 1224; Section D.4.24.20 [showNmzOptions], page 1225.

D.4.24.22 normaliz

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Return: The function applies Normaliz to the matrix input_mat ofe type nmz_type. One can add further pairs "value, type" such as vectors or numerical parameters. See Normaliz manual for possible types and values. The function returns the intmat defined by the file ith suffix gen, provided it has been computed. Otherwise it returns the input_mat.

Note: You will find procedures for many applications of Normaliz in this library, so the explicit call of this procedure may not be necessary.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z),dp;
intmat M[3][2]=3,1,
3,2,
1,3;
normaliz(M,"cone");
↦ 1,1,
↦ 1,2,
↦ 1,3,
↦ 2,1,
↦ 3,1
intmat Inequalities[2][3] = 2,-1,0, // 2x-y >= 0
1, 1,0; //  x+y >= 0
intmat Equation[1][3] = 0,1,-1;    // y = z
intmat Congruence[1][4] = 1,0,0,3;  // x = 0 (3)
normaliz(Inequalities,"inequalities",Equation,"equations",Congruence,"congruences");
↦ 3,-3,-3,
↦ 3,-2,-2,
↦ 3,-1,-1,
↦ 3,0,0,
↦ 3,1,1,
↦ 3,2,2,
↦ 3,3,3,
↦ 3,4,4,
↦ 3,5,5,
↦ 3,6,6
```

See also: Section D.4.24.12 [diagInvariants], page 1219; Section D.4.24.5 [ehrhartRing], page 1213; Section D.4.24.11 [finiteDiagInvariants], page 1218; Section D.4.24.6 [intclMonIdeal], page 1214;

Section D.4.24.1 [intclToricRing], page 1210; Section D.4.24.14 [intersectionValRingIdeals], page 1220; Section D.4.24.13 [intersectionValRings], page 1220; Section D.4.24.2 [normalToricRing], page 1211; Section D.4.24.10 [torusInvariants], page 1217.

D.4.24.23 setNmzExecPath

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: setNmzExecPath(string s); `s` path to the Normaliz executable

Create: `Normaliz::nmz_exec_path` to save the given path `s`

Note: It is not necessary to use this function if the Normaliz executable is in the search path of the system.

Example:

```
LIB "normaliz.lib";
setNmzExecPath("../Normaliz/");
```

See also: Section D.4.24.18 [setNmzOption], page 1224.

D.4.24.24 writeNmzData

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: writeNmzData(intmat M, string nmz_type);
writeNmzData(intmat M, string nmz_type, intmat M2, string nmz_type2, ...);

Create: Creates an input file for Normaliz from the matrix M. The second parameter sets the type. How the matrix is interpreted depends on the type. See the Normaliz documentation for more information.

It is also possible to give more than one pair of matrix and type. In

Note: Needs an explicit filename set. The filename is created from the current filename.
Note that all high level functions in normaliz.lib write and read their data automatically to and from the hard disk so that writeNmzData will hardly ever be used explicitly.

Example:

```
LIB "normaliz.lib";
setNmzFilename("VeryInteresting");
intmat sgr[3][3]=1,2,3,4,5,6,7,8,10;
writeNmzData(sgr,"cone_and_lattice");
int dummy=system("sh","cat VeryInteresting.in");
↦ amb_space auto
↦ cone_and_lattice
↦ [
↦ [1, 2, 3]
↦ [4, 5, 6]
↦ [7, 8, 10]
↦ ]
intmat Inequalities[2][3] = 2,-1,0, // 2x-y >= 0
1, 1,0; //  x+y >= 0
intmat Equation[1][3] = 0,1,-1;    // y = z
intmat Congruence[1][4] = 1,0,0,3;  // x = 0 (3)
writeNmzData(Inequalities,"inequalities",Equation,"equations",Congruence,"congruence
dummy=system("sh","cat VeryInteresting.in");
```

```
↦ amb_space auto
↦ inequalities
↦ [
↦ [2, -1, 0]
↦ [1, 1, 0]
↦ ]
↦ equations
↦ [
↦ [0, 1, -1]
↦ ]
↦ congruences
↦ [
↦ [1, 0, 0, 3]
↦ ]
```

See also: Section D.4.24.25 [readNmzData], page 1228; Section D.4.24.30 [rmNmzFiles], page 1230; Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.26 [setNmzFilename], page 1229.

D.4.24.25 readNmzData

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: readNmzData(string suffix);

Return: Reads an output file of Normaliz containing an integer matrix and returns it as an intmat. For example, this function is useful if one wants to inspect the support hyperplanes. The filename is created from the current filename and the suffix given to the function. In addition to file suffixes, also sup, equ and cgr are allowed. They extract the support hyperplanes, equations and congruences, respectively, from the cst file.

Note: Needs an explicit filename set by setNmzFilename.
Note that all functions in normaliz.lib write and read their data automatically so that readNmzData will usually not be used explicitly.
This function reads only the first matrix in a file! But see su, equ, cgr above.)
It is the responsability of the user to make sure that an output file read by this function has been created for the current input file of Normaliz. Files are not automatically removed before a new computation starts.
Not every output file can be read by this function.

Example:

```
LIB "normaliz.lib";
setNmzFilename("VeryInteresting");
intmat sgr[3][3]=1,2,3,4,5,6,7,8,9;
intmat sgrnormal=normaliz(sgr,"cone");
readNmzData("sup");
↦ -8,7,0,
↦ 2,-1,0
readNmzData("equ");
↦ 1,-2,1
```

See also: Section D.4.24.30 [rmNmzFiles], page 1230; Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.26 [setNmzFilename], page 1229; Section D.4.24.24 [writeNmzData], page 1227.

D.4.24.26 setNmzFilename

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: setNmzFilename(string s);

Create: `Normaliz::nmz_filename` to save the given filename `s`

Note: The function sets the filename for the exchange of data. Unless a path is set by setNmzDataPath, files will be created in the current directory.
If a non-empty filename is set, the files created for and by Normaliz are kept. This is mandatory for the data access functions (see Section D.4.24.24 [writeNmzData], page 1227 and Section D.4.24.25 [readNmzData], page 1228).
Resetting the filename by setNmzFilename("") forces the library to return to deletion of temporary files, but the files created while the filename had been set will not be erased.

Example:

```
     LIB "normaliz.lib";
     setNmzDataPath("examples/");
     setNmzFilename("example1");
     //now the files for the exchange with Normaliz are examples/example1.SUFFIX
```

See also: Section D.4.24.25 [readNmzData], page 1228; Section D.4.24.30 [rmNmzFiles], page 1230; Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.24 [writeNmzData], page 1227.

D.4.24.27 setNmzDataPath

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: setNmzDataPath(string s);

Create: `Normaliz::nmz_data_path` to save the given path `s`

Note: The function sets the path for the exchange of data. By default the files will be created in the current directory.
It seems that Singular cannot use filenames starting with ˜ or `$HOME` in its input/output functions.
You must also avoid path names starting with `/` if you work under Cygwin, since Singular and Normaliz interpret them in different ways.

Example:

```
     LIB "normaliz.lib";
     setNmzDataPath("examples/");
     setNmzFilename("example1");
     //now the files for the exchange with Normalize are examples/example1.SUFFIX
```

See also: Section D.4.24.25 [readNmzData], page 1228; Section D.4.24.30 [rmNmzFiles], page 1230; Section D.4.24.26 [setNmzFilename], page 1229; Section D.4.24.24 [writeNmzData], page 1227.

D.4.24.28 writeNmzPaths

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Create: the file nmz_sing_exec.path where the path to the Normaliz executable is saved
the file nmz_sing_data.path where the directory for the exchange of data is saved

Note: Both files are saved in the current directory. If one of the names has not been defined, the corresponding file is created, but contains nothing.

Example:

```
LIB "normaliz.lib";
setNmzExecPath("../Normaliz/");
writeNmzPaths();
int dummy=system("sh","cat nmz_sing_exec.path");
↦ ../Normaliz/
dummy=system("sh","cat nmz_sing_data.path");
↦
```

See also: Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.23 [setNmzExecPath], page 1227; Section D.4.24.29 [startNmz], page 1230.

D.4.24.29 startNmz

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: startNmz();

Purpose: This function reads the files written by `writeNmzPaths()`, retrieves the path names, and types them on the standard output (as far as they have been set). Thus, once the path names have been stored, a Normaliz session can simply be opened by this function.

Example:

```
LIB "normaliz.lib";
writeNmzPaths();
startNmz();
↦ nmz_exec_path not set
↦ nmz_data_path not set
```

See also: Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.23 [setNmzExecPath], page 1227; Section D.4.24.28 [writeNmzPaths], page 1229.

D.4.24.30 rmNmzFiles

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: rmNmzFiles();

Purpose: This function removes the files created for and by Normaliz, using the last filename specified. It needs an explicit filename set (see Section D.4.24.26 [setNmzFilename], page 1229).

Example:

```
LIB "normaliz.lib";
setNmzFilename("VeryInteresting");
rmNmzFiles();
```

See also: Section D.4.24.25 [readNmzData], page 1228; Section D.4.24.27 [setNmzDataPath], page 1229; Section D.4.24.26 [setNmzFilename], page 1229; Section D.4.24.24 [writeNmzData], page 1227.

D.4.24.31 mons2intmat

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: mons2intmat(ideal I);

Return: Returns the intmat whose rows represent the leading exponents of the (non-zero) elements of I. The length of each row is nvars(basering).

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z),dp;
ideal I=x2,y2,x2yz3;
mons2intmat(I);
↦ 2,0,0,
↦ 0,2,0,
↦ 2,1,3
```

See also: Section D.4.24.32 [intmat2mons], page 1231.

D.4.24.32 intmat2mons

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intmat2mons(intmat M);

Return: an ideal generated by the monomials which correspond to the exponent vectors given by the rows of `M`

Note: The number of variables in the basering `nvars(basering)` has to be at least the number of columns `ncols(M)`, otherwise the function exits with an error. is thrown (see Section 5.1.30 [ERROR], page 177).

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z),dp;
intmat expo_vecs[3][3] =
2,0,0,
0,2,0,
2,1,3;
intmat2mons(expo_vecs);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=x2yz3
```

See also: Section D.4.24.31 [mons2intmat], page 1230.

D.4.24.33 binomials2intmat

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: binomials2intmat(ideal I);

Return: Returns the intmat whose rows represent the exponents of the (non-zero) elements of I which have to be binomials.
The length of each row is nvars(basering).

Example:

```
LIB "normaliz.lib";
ring S = 37,(u,v,w,x,y,z),dp;
ideal I = u2v-xyz, ux2-vyz, uvw-y2z;
binomials2intmat(I);
↦ 2,1,0,-1,-1,-1,
↦ 1,-1,0,2,-1,-1,
↦ 1,1,1,0,-2,-1
```

See also: Section D.4.24.34 [intmat2binomials], page 1232; Section D.4.24.32 [intmat2mons], page 1231; Section D.4.24.31 [mons2intmat], page 1230.

D.4.24.34 intmat2binomials

Procedure from library `normaliz.lib` (see Section D.4.24 [normaliz_lib], page 1209).

Usage: intmat2binomials(intmat M);

Return: an ideal generated by the binomials which correspond to the exponent vectors given by the rows of `M`

Note: The number of variables in the basering `nvars(basering)` has to be at least the number of columns `ncols(M)`, otherwise the function exits with an error is thrown (see Section 5.1.30 [ERROR], page 177). The vector with all entries zero represents 1.

Example:

```
LIB "normaliz.lib";
ring S = 37,(u,v,w,x,y,z),dp;
intmat M[2][6] = 1,0,1,-1,-2,0,  -3,4,5,0,0,1;
intmat2binomials(M);
↦ _[1]=-xy2+uw
↦ _[2]=v4w5z-u3
```

See also: Section D.4.24.33 [binomials2intmat], page 1231.

D.4.25 pointid_lib

Library: pointid.lib

Purpose: Procedures for computing a factorized lex GB of the vanishing ideal of a set of points via the Axis-of-Evil Theorem (M.G. Marinari, T. Mora)

Author: Stefan Steidel, steidel@mathematik.uni-kl.de

Overview: The algorithm of Cerlienco-Mureddu [Marinari M.G., Mora T., A remark on a remark by Macaulay or Enhancing Lazard Structural Theorem. Bull. of the Iranian Math. Soc., 29 (2003), 103-145] associates to each ordered set of points A:={a1,...,as} in K^n, ai:=(ai1,...,ain)
- a set of monomials N and
- a bijection phi: A –> N.
Here I(A):={f in K[x(1),...,x(n)] | f(ai)=0, for all 1<=i<=s} denotes the vanishing ideal of A and N = Mon(x(1),...,x(n)) {LM(f)|f in I(A)} is the set of monomials which do not lie in the leading ideal of I(A) (w.r.t. the lexicographical ordering with x(n)>...>x(1)). N is also called the set of non-monomials of I(A). NOTE: #A = #N and N is a monomial basis of K[x(1..n)]/I(A). In particular, this allows to deduce the set of corner-monomials, i.e. the minimal basis M:={m1,...,mr}, m1<...<mr, of its associated monomial ideal M(I(A)), such that
M(I(A))= {k*mi | k in Mon(x(1),...,x(n)), mi in M},
and (by interpolation) the unique reduced lexicographical Groebner basis G := {f1,...,fr} such that LM(fi)=mi for each i, that is, I(A)=<G>. Moreover, a variation of this algorithm allows to deduce a canonical linear factorization of each element of such a Groebner basis in the sense ot the Axis-of-Evil Theorem by M.G. Marinari and

T. Mora. More precisely, a combinatorial algorithm and interpolation allow to deduce polynomials

y_mdi = x(m) - g_mdi(x(1),...,x(m-1)),
i=1,...,r; m=1,...,n; d in a finite index-set F, satisfying

fi = (product of y_mdi) modulo (f1,...,f(i-1))
where the product runs over all m=1,...,n; and all d in F.

Procedures:

D.4.25.1 nonMonomials

Procedure from library `pointid.lib` (see Section D.4.25 [pointid_lib], page 1232).

Usage: nonMonomials(id); id = <list of vectors> or <list of lists> or <module> or <matrix>.
Let A= {a1,...,as} be a set of points in K^n, ai:=(ai1,...,ain), then A can be given as
- a list of vectors (the ai are vectors) or
- a list of lists (the ai are lists of numbers) or
- a module s.t. the ai are generators or
- a matrix s.t. the ai are columns

Assume: basering must have ordering ip, i.e., be of the form 0,x(1..n),ip; (the first entry of a point belongs to the lex-smallest variable, etc.)

Return: ideal, the non-monomials of the vanishing ideal I(A) of A

Purpose: compute the set of non-monomials Mon(x(1),...,x(n)) \ {LM(f)|f in I(A)} of the vanishing ideal I(A) of the given set of points A in K^n, where K[x(1),...,x(n)] is equipped with the lexicographical ordering induced by x(1)<...<x(n) by using the algorithm of Cerlienco-Mureddu

Example:

```
LIB "pointid.lib";
ring R1 = 0,x(1..3),ip;
vector a1 = [4,0,0];
vector a2 = [2,1,4];
vector a3 = [2,4,0];
vector a4 = [3,0,1];
vector a5 = [2,1,3];
vector a6 = [1,3,4];
vector a7 = [2,4,3];
vector a8 = [2,4,2];
vector a9 = [1,0,2];
list A = a1,a2,a3,a4,a5,a6,a7,a8,a9;
nonMonomials(A);
↦ _[1]=1
↦ _[2]=x(1)
↦ _[3]=x(2)
↦ _[4]=x(1)^2
↦ _[5]=x(3)
↦ _[6]=x(1)^3
↦ _[7]=x(2)*x(3)
↦ _[8]=x(3)^2
↦ _[9]=x(1)*x(2)
```

```
matrix MAT[9][3] = 4,0,0,2,1,4,2,4,0,3,0,1,2,1,3,1,3,4,2,4,3,2,4,2,1,0,2;
MAT = transpose(MAT);
print(MAT);
↦ 4,2,2,3,2,1,2,2,1,
↦ 0,1,4,0,1,3,4,4,0,
↦ 0,4,0,1,3,4,3,2,2
nonMonomials(MAT);
↦ _[1]=1
↦ _[2]=x(1)
↦ _[3]=x(2)
↦ _[4]=x(1)^2
↦ _[5]=x(3)
↦ _[6]=x(1)^3
↦ _[7]=x(2)*x(3)
↦ _[8]=x(3)^2
↦ _[9]=x(1)*x(2)
module MOD = gen(3),gen(2)-2*gen(3),2*gen(1)+2*gen(3),2*gen(2)-2*gen(3),gen(1)+3*gen
print(MOD);
↦ 0,0, 2,0, 1,1,1,
↦ 0,1, 0,2, 0,1,1,
↦ 1,-2,2,-2,3,3,1
nonMonomials(MOD);
↦ _[1]=1
↦ _[2]=x(2)
↦ _[3]=x(1)
↦ _[4]=x(2)^2
↦ _[5]=x(1)^2
↦ _[6]=x(1)*x(2)
↦ _[7]=x(3)
ring R2 = 0,x(1..2),ip;
list l1 = 0,0;
list l2 = 0,1;
list l3 = 2,0;
list l4 = 0,2;
list l5 = 1,0;
list l6 = 1,1;
list L = l1,l2,l3,l4,l5,l6;
nonMonomials(L);
↦ _[1]=1
↦ _[2]=x(2)
↦ _[3]=x(1)
↦ _[4]=x(2)^2
↦ _[5]=x(1)^2
↦ _[6]=x(1)*x(2)
```

D.4.25.2 cornerMonomials

Procedure from library `pointid.lib` (see Section D.4.25 [pointid_lib], page 1232).

Usage: cornerMonomials(N); N ideal

Assume: N is given by monomials satisfying the condition that if a monomial is in N then any of its factors is in N (N is then called an order ideal)

Return: ideal, the corner-monomials of the order ideal N
The corner-monomials are the leading monomials of an ideal I s.t. N is a basis of basering/I.

Note: In our applications, I is the vanishing ideal of a finte set of points.

Example:

```
LIB "pointid.lib";
ring R = 0,x(1..3),ip;
poly n1 = 1;
poly n2 = x(1);
poly n3 = x(2);
poly n4 = x(1)^2;
poly n5 = x(3);
poly n6 = x(1)^3;
poly n7 = x(2)*x(3);
poly n8 = x(3)^2;
poly n9 = x(1)*x(2);
ideal N = n1,n2,n3,n4,n5,n6,n7,n8,n9;
cornerMonomials(N);
↦ _[1]=x(1)^4
↦ _[2]=x(1)^2*x(2)
↦ _[3]=x(2)^2
↦ _[4]=x(1)*x(3)
↦ _[5]=x(2)*x(3)^2
↦ _[6]=x(3)^3
```

D.4.25.3 facGBIdeal

Procedure from library `pointid.lib` (see Section D.4.25 [pointid_lib], page 1232).

Usage: facGBIdeal(id); id = <list of vectors> or <list of lists> or <module> or <matrix>.
Let A= {a1,...,as} be a set of points in K^n, ai:=(ai1,...,ain), then A can be given as
- a list of vectors (the ai are vectors) or
- a list of lists (the ai are lists of numbers) or
- a module s.t. the ai are generators or
- a matrix s.t. the ai are columns

Assume: basering must have ordering ip, i.e., be of the form 0,x(1..n),ip; (the first entry of a point belongs to the lex-smallest variable, etc.)

Return: a list where the first entry contains the Groebner basis G of I(A) and the second entry contains the linear factors of each element of G

Note: combinatorial algorithm due to the Axis-of-Evil Theorem of M.G. Marinari, T. Mora

Example:

```
LIB "pointid.lib";
ring R = 0,x(1..3),ip;
vector a1 = [4,0,0];
vector a2 = [2,1,4];
vector a3 = [2,4,0];
vector a4 = [3,0,1];
vector a5 = [2,1,3];
vector a6 = [1,3,4];
vector a7 = [2,4,3];
vector a8 = [2,4,2];
vector a9 = [1,0,2];
list A = a1,a2,a3,a4,a5,a6,a7,a8,a9;
facGBIdeal(A);
↦ [1]:
↦    _[1]=x(1)^4-10*x(1)^3+35*x(1)^2-50*x(1)+24
↦    _[2]=x(1)^2*x(2)-3*x(1)*x(2)+2*x(2)
↦    _[3]=x(2)^2-1/2*x(1)^2*x(2)-1/2*x(1)*x(2)-2*x(2)+2*x(1)^3-16*x(1)^2+38\
   *x(1)-24
↦    _[4]=x(1)*x(3)-2*x(3)-2/3*x(1)*x(2)+4/3*x(2)+1/6*x(1)^3-1/2*x(1)^2-5/3\
   *x(1)+4
↦    _[5]=x(2)*x(3)^2-4*x(3)^2-2/3*x(2)^2*x(3)-5/6*x(1)^3*x(2)*x(3)+41/6*x(\
   1)^2*x(2)*x(3)-16*x(1)*x(2)*x(3)+23/3*x(2)*x(3)+10/3*x(1)^3*x(3)-82/3*x(1\
   )^2*x(3)+64*x(1)*x(3)-20*x(3)+2*x(2)^2+5/2*x(1)^3*x(2)-41/2*x(1)^2*x(2)+4\
   8*x(1)*x(2)-32*x(2)-10*x(1)^3+82*x(1)^2-192*x(1)+96
↦    _[6]=x(3)^3+4/3*x(2)*x(3)^2-5/6*x(1)^3*x(3)^2+35/6*x(1)^2*x(3)^2-9*x(1\
   )*x(3)^2-9*x(3)^2-20/3*x(2)*x(3)+25/6*x(1)^3*x(3)-175/6*x(1)^2*x(3)+45*x(\
   1)*x(3)+26*x(3)+8*x(2)-5*x(1)^3+35*x(1)^2-54*x(1)-24
↦ [2]:
↦    [1]:
↦       _[1]=x(1)-4
↦       _[2]=x(1)-2
↦       _[3]=x(1)-3
↦       _[4]=x(1)-1
↦    [2]:
↦       _[1]=x(1)-2
↦       _[2]=x(1)-1
↦       _[3]=x(2)
↦    [3]:
↦       _[1]=x(2)-4*x(1)+4
↦       _[2]=2*x(2)-x(1)^2+7*x(1)-12
↦    [4]:
↦       _[1]=x(1)-2
↦       _[2]=6*x(3)-4*x(2)+x(1)^2-x(1)-12
↦    [5]:
↦       _[1]=x(2)-4
↦       _[2]=x(3)-3
↦       _[3]=6*x(3)-4*x(2)-5*x(1)^3+41*x(1)^2-96*x(1)+48
↦    [6]:
↦       _[1]=x(3)-2
↦       _[2]=x(3)-3
↦       _[3]=6*x(3)+8*x(2)-5*x(1)^3+35*x(1)^2-54*x(1)-24
matrix MAT[9][3] = 4,0,0,2,1,4,2,4,0,3,0,1,2,1,3,1,3,4,2,4,3,2,4,2,1,0,2;
MAT = transpose(MAT);
print(MAT);
↦ 4,2,2,3,2,1,2,2,1,
↦ 0,1,4,0,1,3,4,4,0,
↦ 0,4,0,1,3,4,3,2,2
facGBIdeal(MAT);
↦ [1]:
↦    _[1]=x(1)^4-10*x(1)^3+35*x(1)^2-50*x(1)+24
↦    _[2]=x(1)^2*x(2)-3*x(1)*x(2)+2*x(2)
↦    _[3]=x(2)^2-1/2*x(1)^2*x(2)-1/2*x(1)*x(2)-2*x(2)+2*x(1)^3-16*x(1)^2+38\
  *x(1)-24
↦    _[4]=x(1)*x(3)-2*x(3)-2/3*x(1)*x(2)+4/3*x(2)+1/6*x(1)^3-1/2*x(1)^2-5/3\
  *x(1)+4
↦    _[5]=x(2)*x(3)^2-4*x(3)^2-2/3*x(2)^2*x(3)-5/6*x(1)^3*x(2)*x(3)+41/6*x(\
  1)^2*x(2)*x(3)-16*x(1)*x(2)*x(3)+23/3*x(2)*x(3)+10/3*x(1)^3*x(3)-82/3*x(1\
  )^2*x(3)+64*x(1)*x(3)-20*x(3)+2*x(2)^2+5/2*x(1)^3*x(2)-41/2*x(1)^2*x(2)+4\
  8*x(1)*x(2)-32*x(2)-10*x(1)^3+82*x(1)^2-192*x(1)+96
↦    _[6]=x(3)^3+4/3*x(2)*x(3)^2-5/6*x(1)^3*x(3)^2+35/6*x(1)^2*x(3)^2-9*x(1\
  )*x(3)^2-9*x(3)^2-20/3*x(2)*x(3)+25/6*x(1)^3*x(3)-175/6*x(1)^2*x(3)+45*x(\
  1)*x(3)+26*x(3)+8*x(2)-5*x(1)^3+35*x(1)^2-54*x(1)-24
↦ [2]:
↦    [1]:
↦       _[1]=x(1)-4
↦       _[2]=x(1)-2
↦       _[3]=x(1)-3
↦       _[4]=x(1)-1
↦    [2]:
↦       _[1]=x(1)-2
↦       _[2]=x(1)-1
↦       _[3]=x(2)
↦    [3]:
↦       _[1]=x(2)-4*x(1)+4
↦       _[2]=2*x(2)-x(1)^2+7*x(1)-12
↦    [4]:
↦       _[1]=x(1)-2
↦       _[2]=6*x(3)-4*x(2)+x(1)^2-x(1)-12
↦    [5]:
↦       _[1]=x(2)-4
↦       _[2]=x(3)-3
↦       _[3]=6*x(3)-4*x(2)-5*x(1)^3+41*x(1)^2-96*x(1)+48
↦    [6]:
↦       _[1]=x(3)-2
↦       _[2]=x(3)-3
↦       _[3]=6*x(3)+8*x(2)-5*x(1)^3+35*x(1)^2-54*x(1)-24
module MOD = gen(3),gen(2)-2*gen(3),2*gen(1)+2*gen(3),2*gen(2)-2*gen(3),gen(1)+3*gen(
print(MOD);
↦ 0,0, 2,0, 1,1,1,
↦ 0,1, 0,2, 0,1,1,
↦ 1,-2,2,-2,3,3,1
facGBIdeal(MOD);
↦ [1]:
↦    _[1]=x(1)^3-3*x(1)^2+2*x(1)
↦    _[2]=x(1)^2*x(2)-x(1)*x(2)
↦    _[3]=x(1)*x(2)^2-x(1)*x(2)
↦    _[4]=x(2)^3-3*x(2)^2+2*x(2)
↦    _[5]=x(1)*x(3)-x(3)-3/2*x(1)*x(2)^2+3/2*x(2)^2+9/2*x(1)*x(2)-9/2*x(2)-\
  1/2*x(1)^2-1/2*x(1)+1
↦    _[6]=x(2)*x(3)-x(3)+3/2*x(2)^2+3/2*x(1)^2*x(2)-7/2*x(1)*x(2)-5/2*x(2)-\
  3/2*x(1)^2+7/2*x(1)+1
↦    _[7]=x(3)^2+3*x(1)^2*x(3)-8*x(1)*x(3)+x(3)-3*x(1)^2+8*x(1)-2
↦ [2]:
↦    [1]:
↦       _[1]=x(1)
↦         _[2]=x(1)-2
↦         _[3]=x(1)-1
↦      [2]:
↦         _[1]=x(1)
↦         _[2]=x(1)-1
↦         _[3]=x(2)
↦      [3]:
↦         _[1]=x(1)
↦         _[2]=x(2)-1
↦         _[3]=x(2)
↦      [4]:
↦         _[1]=x(2)-2
↦         _[2]=x(2)-1
↦         _[3]=x(2)
↦      [5]:
↦         _[1]=x(1)-1
↦         _[2]=2*x(3)-3*x(2)^2+9*x(2)-x(1)-2
↦      [6]:
↦         _[1]=x(2)-1
↦         _[2]=2*x(3)+3*x(2)+3*x(1)^2-7*x(1)-2
↦      [7]:
↦         _[1]=x(3)-1
↦         _[2]=x(3)+3*x(1)^2-8*x(1)+2
list l1 = 0,0,1;
list l2 = 0,1,-2;
list l3 = 2,0,2;
list l4 = 0,2,-2;
list l5 = 1,0,3;
list l6 = 1,1,3;
list L = l1,l2,l3,l4,l5,l6;
facGBIdeal(L);
↦ [1]:
↦    _[1]=x(1)^3-3*x(1)^2+2*x(1)
↦    _[2]=x(1)^2*x(2)-x(1)*x(2)
↦    _[3]=x(1)*x(2)^2-x(1)*x(2)
↦    _[4]=x(2)^3-3*x(2)^2+2*x(2)
↦    _[5]=x(3)-3/2*x(2)^2-3*x(1)*x(2)+9/2*x(2)+3/2*x(1)^2-7/2*x(1)-1
↦ [2]:
↦      [1]:
↦         _[1]=x(1)
↦         _[2]=x(1)-2
↦         _[3]=x(1)-1
↦      [2]:
↦         _[1]=x(1)
↦         _[2]=x(1)-1
↦         _[3]=x(2)
↦      [3]:
↦         _[1]=x(1)
↦         _[2]=x(2)-1
↦         _[3]=x(2)
↦      [4]:
↦         _[1]=x(2)-2
↦         _[2]=x(2)-1
↦        _[3]=x(2)
↦     [5]:
↦        _[1]=2*x(3)-3*x(2)^2-6*x(1)*x(2)+9*x(2)+3*x(1)^2-7*x(1)-2
```

D.4.26 primdec_lib

Library: primdec.lib

Purpose: Primary Decomposition and Radical of Ideals

Authors: Gerhard Pfister, pfister@mathematik.uni-kl.de (GTZ)
Wolfram Decker, decker@math.uni-sb.de (SY)
Hans Schoenemann, hannes@mathematik.uni-kl.de (SY)
Santiago Laplagne, slaplagn@dm.uba.ar (GTZ)

Overview: Algorithms for primary decomposition based on the ideals of Gianni, Trager and Zacharias (implementation by Gerhard Pfister), respectively based on the ideas of Shimoyama and Yokoyama (implementation by Wolfram Decker and Hans Schoenemann).
The procedures are implemented to be used in characteristic 0.
They also work in positive characteristic >> 0.
In small characteristic and for algebraic extensions, primdecGTZ may not terminate.
Algorithms for the computation of the radical based on the ideas of Krick, Logar, Laplagne and Kemper (implementation by Gerhard Pfister and Santiago Laplagne).
They work in any characteristic.
Baserings must have a global ordering and no quotient ideal. Exceptions: primdecGTZ, absPrimdecGTZ, minAssGTZ, primdecSY, minAssChar, radical accept non-global ordering.

Procedures: See also: Section D.4.27 [primdecint_lib], page 1252.

D.4.26.1 Ann

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: Ann(M); M module

Return: ideal, the annihilator of coker(M)

Note: The output is the ideal of all elements a of the basering R such that a * R^m is contained in M (m=number of rows of M).

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
module M = x2-y2,z3;
Ann(M);
↦ _[1]=z3
↦ _[2]=x2-y2
M = [1,x2],[y,x];
Ann(M);
↦ _[1]=x2y-x
qring Q=std(xy-1);
module M=imap(r,M);
Ann(M);
↦ _[1]=0
```

D.4.26.2 primdecGTZ

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: primdecGTZ(I); I ideal

Return: a list pr of primary ideals and their associated primes for a proper ideal I, otherwise pr = list(list(ideal(1), ideal(1))

pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

Note: - Algorithm of Gianni/Trager/Zacharias.
- Designed for characteristic 0, works also in char k > 0, if it terminates (may result in an infinite loop in small characteristic!) - For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring
- For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
list pr = primdecGTZ(i);
pr;
↦ [1]:
↦    [1]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦ [2]:
↦    [1]:
↦       _[1]=z6+4z3+4
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z3+2
↦       _[2]=y-z2
```

D.4.26.3 primdecGTZE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: primdecGTZE(I); i ideal

Return: a list pr of primary ideals and their associated primes for a proper ideal, and an empty list for the unit ideal.

pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

Note: - Algorithm of Gianni/Trager/Zacharias.
- Designed for characteristic 0, works also in char k > 0, if it terminates (may result in an infinite loop in small characteristic!) - For local orderings, the result is considered

in the localization of the polynomial ring, not in the power series ring
- For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal I = p*q^2,y-z2;
list pr = primdecGTZE(I);
pr;
↦ [1]:
↦    [1]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦ [2]:
↦    [1]:
↦       _[1]=z6+4z3+4
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z3+2
↦       _[2]=y-z2
ideal J = 1;
list prempty = primdecGTZE(J);
prempty;
↦ empty list
```

D.4.26.4 primdecSY

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: primdecSY(I, c); I ideal, c int (optional)

Return: a list pr of primary ideals and their associated primes for proper ideal I, otherwise pr[1] is list(ideal(1),ideal(1))'

pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

Note: Algorithm of Shimoyama/Yokoyama.

if c=0, the given ordering of the variables is used,
if c=1, minAssChar tries to use an optimal ordering (default),
if c=2, minAssGTZ is used,
if c=3, minAssGTZ and facstd are used.

For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring.
For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as third argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
list pr = primdecSY(i);
pr;
↦ [1]:
↦    [1]:
↦       _[1]=z6+4z3+4
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z3+2
↦       _[2]=y-z2
↦ [2]:
↦    [1]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z2+1
↦       _[2]=y+1
```

D.4.26.5 primdecSYE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: primdecSYE(I, c); I ideal, c int (optional)

Return: a list pr of primary ideals and their associated primes:

pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

If I is the unit ideal returns an empty list.

Note: Algorithm of Shimoyama/Yokoyama.

if c=0, the given ordering of the variables is used,
if c=1, minAssChar tries to use an optimal ordering (default),
if c=2, minAssGTZ is used,
if c=3, minAssGTZ and facstd are used.

For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring.
For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as third argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal I = p*q^2,y-z2;
list pr = primdecSYE(I);
pr;
↦ [1]:
↦    [1]:
↦       _[1]=z6+4z3+4
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z3+2
↦       _[2]=y-z2
↦ [2]:
↦    [1]:
↦       _[1]=z2+1
↦       _[2]=y-z2
↦    [2]:
↦       _[1]=z2+1
↦       _[2]=y+1
ideal J = x;
list prUnit = primdecSYE(J);
prUnit;
↦ [1]:
↦    [1]:
↦       _[1]=x
↦    [2]:
↦       _[1]=x
```

D.4.26.6 minAssGTZ

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: minAssGTZ(I[, l]); I ideal, l list (optional)
Optional parameters in list l (can be entered in any order):
0, "facstd" -> uses facstd to first decompose the ideal (default)
1, "noFacstd" -> does not use facstd
"GTZ" -> the original algorithm by Gianni, Trager and Zacharias is used
"SL" -> GTZ algorithm with modificiations by Laplagne is used (default)

Return: a list, the minimal associated prime ideals of proper ideal I, otherwise ideal(1)

Note: - Designed for characteristic 0, works also in char k > 0 based on an algorithm of Yokoyama
- For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring
- For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
list pr = minAssGTZ(i);
pr;
↦ [1]:
↦    _[1]=z3+2
↦    _[2]=-z2+y
↦ [2]:
↦    _[1]=z2+1
↦    _[2]=-z2+y
```

D.4.26.7 minAssGTZE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: minAssGTZE(I[, l]); I ideal, l list (optional)
Optional parameters in list l (can be entered in any order):
0, "facstd" -> uses facstd to first decompose the ideal (default)
1, "noFacstd" -> does not use facstd
"GTZ" -> the original algorithm by Gianni, Trager and Zacharias is used
"SL" -> GTZ algorithm with modificiations by Laplagne is used (default)

Return: a list, the minimal associated prime ideals of I.

Note: - Designed for characteristic 0, works also in char k > 0 based on an algorithm of Yokoyama
- For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring
- For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal I = p*q^2,y-z2;
list pr = minAssGTZE(I);
pr;
↦ [1]:
↦    _[1]=z3+2
↦    _[2]=-z2+y
↦ [2]:
↦    _[1]=z2+1
↦    _[2]=-z2+y
ideal J = 1;
list prempty = minAssGTZE(J);
prempty;
↦ empty list
```

D.4.26.8 minAssChar

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: minAssChar(I[,c]); i ideal, c int (optional).

Return: list, the minimal associated prime ideals of I. If I is the unit ideal returns list(ideal(1))

Note: If c=0, the given ordering of the variables is used.
Otherwise, the system tries to find an optimal ordering, which in some cases may considerably speed up the algorithm.
For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring
For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as third argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
list pr = minAssChar(i);
pr;
↦ [1]:
↦    _[1]=y+1
↦    _[2]=z2+1
↦ [2]:
↦    _[1]=z2-y
↦    _[2]=yz+2
↦    _[3]=y2+2z
```

D.4.26.9 minAssCharE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: minAssCharE(I[,c]); i ideal, c int (optional).

Return: list, the minimal associated prime ideals of I. If I is the unit ideal returns an empty list.

Note: If c=0, the given ordering of the variables is used.
Otherwise, the system tries to find an optimal ordering, which in some cases may considerably speed up the algorithm.
For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring
For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as third argument

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal I = p*q^2,y-z2;
list pr = minAssCharE(I);
pr;
↦ [1]:
↦    _[1]=y+1
↦    _[2]=z2+1
↦ [2]:
↦    _[1]=z2-y
↦    _[2]=yz+2
↦    _[3]=y2+2z
ideal J = 5;
list prempty = minAssCharE(J);
prempty;
↦ empty list
```

D.4.26.10 testPrimary

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: testPrimary(pr,k); pr a list, k an ideal.

Assume: pr is the result of primdecGTZ(k) or primdecSY(k).

Return: int, 1 if the intersection of the ideals in pr is k, 0 if not

Example:

```
LIB "primdec.lib";
ring  r = 32003,(x,y,z),dp;
poly  p = z2+1;
poly  q = z4+2;
ideal i = p^2*q^3,(y-z3)^3,(x-yz+z4)^4;
list pr = primdecGTZ(i);
testPrimary(pr,i);
↦ 1
```

D.4.26.11 testPrimaryE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: testPrimaryE(pr,k); pr a list, k an ideal.

Assume: pr is the result of a primary decomposition and may be empty (for the unit ideal)

Return: int, 1 if the intersection of the ideals in pr is k, 0 if not

Example:

```
LIB "primdec.lib";
ring  r = 32003,(x,y,z),dp;
poly  p = z2+1;
poly  q = z4+2;
ideal i = p^2*q^3,(y-z3)^3,(x-yz+z4)^4;
list pr = primdecGTZ(i);
testPrimaryE(pr,i);
↦ 1
```

D.4.26.12 radical

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: radical(I[, l]); I ideal, l list (optional)
Optional parameters in list l (can be entered in any order):
0, "fullRad" -> full radical is computed (default)
1, "equiRad" -> equiRadical is computed
"KL" -> Krick/Logar algorithm is used
"SL" -> modifications by Laplagne are used (default)
"facstd" -> uses facstd to first decompose the ideal (default for non homogeneous ideals)
"noFacstd" -> does not use facstd (default for homogeneous ideals)

Return: ideal, the radical of I (or the equiradical if required in the input parameters)

Note: A combination of the algorithms of Krick/Logar (with modifications by Laplagne) and Kemper is used. Works also in positive characteristic (Kempers algorithm).

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
ideal pr = radical(i);
pr;
↦ pr[1]=z2-y
↦ pr[2]=y2z+z3+2z2+2
```

D.4.26.13 radicalEHV

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: radicalEHV(i); i ideal.

Return: ideal, the radical of i.

Note: Uses the algorithm of Eisenbud/Huneke/Vasconcelos, which reduces the computation to the complete intersection case, by taking, in the general case, a generic linear combination of the input.
Works only in characteristic 0 or p large.

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
ideal pr= radicalEHV(i);
pr;
↦ pr[1]=z2-y
↦ pr[2]=y2z+yz+2y+2
↦ pr[3]=y3+y2+2yz+2z
```

D.4.26.14 equiRadical

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: equiRadical(I); I ideal

Return: ideal, intersection of associated primes of I of maximal dimension.

Note: A combination of the algorithms of Krick/Logar (with modifications by Laplagne) and Kemper is used. Works also in positive characteristic (Kempers algorithm).

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
ideal pr= equiRadical(i);
pr;
↦ pr[1]=z2-y
↦ pr[2]=y2z+z3+2z2+2
```

D.4.26.15 prepareAss

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: prepareAss(I); I ideal

Return: list of radicals of the equidimensional components of I

Note: Uses algorithm of Eisenbud/Huneke/Vasconcelos.

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
list pr = prepareAss(i);
pr;
↦ [1]:
↦    _[1]=z2-y
↦    _[2]=y2z+z3+2z2+2
```

D.4.26.16 equidim

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: equidim(I) or equidim(I,1) ; I ideal

Return: list of equidimensional ideals a[1],...,a[s] with:
- a[s] the equidimensional locus of I, i.e. the intersection of the primary ideals of dimension of I, except I is unit ideal. - a[1],...,a[s-1] the lower dimensional equidimensional loci. If I is the unit ideal, a list containing the unit ideal as a[1] is returned.

Note: An embedded component q (primary ideal) of I can be replaced in the decomposition by a primary ideal q1 with the same radical as q.
`equidim(I,1)` uses the algorithm of Eisenbud/Huneke/Vasconcelos.

Example:

```
LIB "primdec.lib";
ring  r = 32003,(x,y,z),dp;
ideal i = intersect(ideal(z),ideal(x,y),ideal(x2,z2),ideal(x5,y5,z5));
equidim(i);
↦ [1]:
↦    _[1]=z4
↦    _[2]=y5
↦    _[3]=x5
↦    _[4]=x3z3
↦    _[5]=x4y4
↦ [2]:
↦    _[1]=yz
↦    _[2]=xz
↦    _[3]=x2
↦ [3]:
↦    _[1]=z
```

D.4.26.17 equidimMax

Procedure from library primdec.lib (see Section D.4.26 [primdec_lib], page 1239).

Usage: equidimMax(i); i ideal

Return: ideal of equidimensional locus (of maximal dimension) of i.

Example:

```
LIB "primdec.lib";
ring  r = 32003,(x,y,z),dp;
ideal i = intersect(ideal(z),ideal(x,y),ideal(x2,z2),ideal(x5,y5,z5));
equidimMax(i);
↦ _[1]=z
```

D.4.26.18 equidimMaxEHV

Procedure from library primdec.lib (see Section D.4.26 [primdec_lib], page 1239).

Usage: equidimMaxEHV(I); I ideal

Return: ideal, the equidimensional component (of maximal dimension) of I.

Note: Uses algorithm of Eisenbud, Huneke and Vasconcelos.

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),dp;
ideal i=intersect(ideal(z),ideal(x,y),ideal(x2,z2),ideal(x5,y5,z5));
equidimMaxEHV(i);
↦ _[1]=z
```

D.4.26.19 zerodec

Procedure from library primdec.lib (see Section D.4.26 [primdec_lib], page 1239).

Usage: zerodec(I); I ideal

Assume: I is zero-dimensional, the characteristic of the ground field is 0

Return: list of primary ideals, the zero-dimensional decomposition of I

Note: The algorithm (of Monico), works well only for a small total number of solutions (vdim(std(I)) should be < 100) and without parameters. In practice, it works also in large characteristic p>0 but may fail for small p.
If printlevel > 0 (default = 0) additional information is displayed.

Example:

```
LIB "primdec.lib";
ring r  = 0,(x,y),dp;
ideal i = x2-2,y2-2;
list pr = zerodec(i);
pr;
↦ [1]:
↦    _[1]=y2-2
↦    _[2]=xy-2
↦    _[3]=x2-2
↦ [2]:
↦    _[1]=y2-2
↦    _[2]=xy+2
↦    _[3]=x2-2
```

D.4.26.20 absPrimdecGTZ

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: absPrimdecGTZ(I); I ideal

Assume: Ground field has characteristic 0.

Return: a ring containing two lists: `absolute_primes`, the absolute prime components of I, and `primary_decomp`, the output of `primdecGTZ(I)`. The list absolute_primes has to be interpreted as follows: each entry describes a class of conjugated absolute primes,

absolute_primes[i][1] the absolute prime component,
absolute_primes[i][2] the number of conjugates.

The first entry of `absolute_primes[i][1]` is the minimal polynomial of a minimal finite field extension over which the absolute prime component is defined.
For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring. For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Note: Algorithm of Gianni/Trager/Zacharias combined with the `absFactorize` command.

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal i = p*q^2,y-z2;
def S = absPrimdecGTZ(i);
↦
↦ // 'absPrimdecGTZ' created a ring, in which two lists absolute_primes (th\
   e
↦ // absolute prime components) and primary_decomp (the primary and prime
↦ // components over the current basering) are stored.
↦ // To access the list of absolute prime components, type (if the name S w\
   as
↦ // assigned to the return value):
↦         setring S; absolute_primes;
setring S;
absolute_primes;
↦ [1]:
↦    [1]:
↦       _[1]=a2+1
↦       _[2]=z-a
↦       _[3]=y+1
↦    [2]:
↦       2
↦ [2]:
↦    [1]:
↦       _[1]=a3+2
↦       _[2]=z-a
↦       _[3]=y-a2
↦    [2]:
↦       3
```

See also: Section D.4.1.1 [absFactorize], page 1001; Section D.4.26.2 [primdecGTZ], page 1240.

D.4.26.21 absPrimdecGTZE

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Usage: absPrimdecGTZE(I); I ideal

Assume: Ground field has characteristic 0.

Return: a ring containing two lists: `absolute_primes`, the absolute prime components of I, and `primary_decomp`, the output of `primdecGTZ(I)`. Will fail for unit ideal. The list absolute_primes has to be interpreted as follows: each entry describes a class of conjugated absolute primes,

absolute_primes[i][1]	the absolute prime component,
absolute_primes[i][2]	the number of conjugates.

The first entry of `absolute_primes[i][1]` is the minimal polynomial of a minimal finite field extension over which the absolute prime component is defined.
For local orderings, the result is considered in the localization of the polynomial ring, not in the power series ring. For local and mixed orderings, the decomposition in the corresponding global ring is returned if the string 'global' is specified as second argument

Note: Algorithm of Gianni/Trager/Zacharias combined with the `absFactorize` command.

Example:

```
LIB "primdec.lib";
ring  r = 0,(x,y,z),lp;
poly  p = z2+1;
poly  q = z3+2;
ideal I = p*q^2,y-z2;
def S = absPrimdecGTZE(I);
↦
↦ // 'absPrimdecGTZ' created a ring, in which two lists absolute_primes (th\
   e
↦ // absolute prime components) and primary_decomp (the primary and prime
↦ // components over the current basering) are stored.
↦ // To access the list of absolute prime components, type (if the name S w\
   as
↦ // assigned to the return value):
↦         setring S; absolute_primes;
setring S;
absolute_primes;
↦ [1]:
↦    [1]:
↦       _[1]=a2+1
↦       _[2]=z-a
↦       _[3]=y+1
↦    [2]:
↦       2
↦ [2]:
↦    [1]:
↦       _[1]=a3+2
↦       _[2]=z-a
↦       _[3]=y-a2
↦    [2]:
↦       3
```

See also: Section D.4.1.1 [absFactorize], page 1001; Section D.4.26.2 [primdecGTZ], page 1240.

D.4.26.22 sep

Procedure from library `primdec.lib` (see Section D.4.26 [primdec_lib], page 1239).

Example:

```
LIB "primdec.lib";
ring R=(5,t,s),(x,y,z),dp;
poly f=(x^25-t*x^5+t)*(x^3+s);
sep(f,1);
↦ [1]:
↦    x8+(s5)*x5+(-t)*x4+(t)*x3+(-ts5)*x+(ts5)
↦ [2]:
↦    1
```

D.4.27 primdecint_lib

Library: primdecint.lib

Purpose: primary decomposition of an ideal in the polynomial ring over the integers

Authors: G. Pfister pfister@mathematik.uni-kl.de
A. Sadiq afshanatiq@gmail.com
S. Steidel steidel@mathematik.uni-kl.de

Overview: A library for computing the primary decomposition of an ideal in the polynomial ring over the integers, Z[x_1,...,x_n].
The first procedure 'primdecZ' can be used in parallel. The coefficients must always be ZZ.

Reference: Pfister, Sadiq, Steidel , "An Algorithm for primary decomposition in polynomial rings over the integers" , arXiv:1008.2074

Procedures: See also: Section D.4.26 [primdec_lib], page 1239.

D.4.27.1 primdecZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Usage: primdecZ(I[, n]); I ideal, n integer (number of processors)

Note: If size(#) > 0, then #[1] is the number of available processors for the computation. The coefficients must be ZZ.

Return: a list pr of primary ideals and their associated primes:
pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2);
I=intersectZ(I,I3);
I=intersectZ(I,I4);
I=intersectZ(I,I5);
I=intersectZ(I,I6);
primdecZ(I);
↦ [1]:
↦    [1]:
↦       _[1]=d5
↦       _[2]=c5
↦    [2]:
↦       _[1]=d
↦       _[2]=c
↦ [2]:
↦    [1]:
↦       _[1]=a2
↦       _[2]=b4
↦    [2]:
↦       _[1]=b
↦       _[2]=a
↦ [3]:
↦    [1]:
↦       _[1]=2
↦       _[2]=c5
↦    [2]:
↦       _[1]=2
↦       _[2]=c
↦ [4]:
↦    [1]:
↦       _[1]=3
↦    [2]:
↦       _[1]=3
↦ [5]:
↦    [1]:
↦       _[1]=13
↦       _[2]=b4
↦    [2]:
↦       _[1]=13
↦       _[2]=b
↦ [6]:
↦    [1]:
↦       _[1]=17
↦       _[2]=a2
↦    [2]:
↦       _[1]=17
↦       _[2]=a
↦ [7]:
↦    [1]:
↦       _[1]=17
↦       _[2]=d15
↦       _[3]=c15
↦       _[4]=b15
↦          _[5]=a15
↦       [2]:
↦          _[1]=17
↦          _[2]=d
↦          _[3]=c
↦          _[4]=b
↦          _[5]=a
↦    [8]:
↦       [1]:
↦          _[1]=9
↦          _[2]=3d5
↦          _[3]=d10
↦       [2]:
↦          _[1]=3
↦          _[2]=d
ideal J=intersectZ(ideal(17,a),ideal(17,a2,b));
primdecZ(J);
↦    [1]:
↦       [1]:
↦          _[1]=17
↦          _[2]=a
↦       [2]:
↦          _[1]=17
↦          _[2]=a
↦    [2]:
↦       [1]:
↦          _[1]=17
↦          _[2]=b
↦          _[3]=a2
↦       [2]:
↦          _[1]=17
↦          _[2]=b
↦          _[3]=a
ideal K=intersectZ(ideal(9,a+3),ideal(9,b+3));
primdecZ(K);
↦    [1]:
↦       [1]:
↦          _[1]=9
↦          _[2]=b+3
↦       [2]:
↦          _[1]=3
↦          _[2]=b
↦    [2]:
↦       [1]:
↦          _[1]=9
↦          _[2]=a+3
↦       [2]:
↦          _[1]=3
↦          _[2]=a
```

D.4.27.2 primdecZM

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint lib], page 1252).

Usage: primdecZM(N); N module
The coefficients must be ZZ.

Return: a list pr of primary modules and their associated primes:
pr[i][1] the i-th primary component,
pr[i][2] the i-th prime component.

Example:

```
LIB "primdecint.lib";
ring R=integer,(x,y),(c,lp);
module N=[0,0,xy2-x2-xy],[0,y,x],[0,x,2xy-x],[x,0,-xy],[0,0,18x];
primdecZM(N);
↦ [1]:
↦    [1]:
↦       _[1]=[0,0,x]
↦       _[2]=[0,1]
↦       _[3]=[x]
↦    [2]:
↦       _[1]=x
↦ [2]:
↦    [1]:
↦       _[1]=[0,0,y3]
↦       _[2]=[0,0,18x]
↦       _[3]=[0,0,xy2]
↦       _[4]=[0,0,x2+xy]
↦       _[5]=[0,y,x]
↦       _[6]=[0,x,2xy-x]
↦       _[7]=[y3]
↦       _[8]=[x,0,-xy]
↦    [2]:
↦       _[1]=y
↦       _[2]=x
```

D.4.27.3 minAssZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Usage: minAssZ(I); I ideal
The coefficients must be ZZ.

Return: a list pr of associated primes:

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2);
I=intersectZ(I,I3);
I=intersectZ(I,I4);
I=intersectZ(I,I5);
```

```
I=intersectZ(I,I6);
minAssZ(I);
↦ [1]:
↦    _[1]=d
↦    _[2]=c
↦ [2]:
↦    _[1]=b
↦    _[2]=a
↦ [3]:
↦    _[1]=2
↦    _[2]=c
↦ [4]:
↦    _[1]=3
↦ [5]:
↦    _[1]=13
↦    _[2]=b
↦ [6]:
↦    _[1]=17
↦    _[2]=a
ideal J=intersectZ(ideal(17,a),ideal(17,a2,b));
minAssZ(J);
↦ [1]:
↦    _[1]=17
↦    _[2]=a
ideal K=intersectZ(ideal(9,a+3),ideal(9,b+3));
minAssZ(K);
↦ [1]:
↦    _[1]=3
↦    _[2]=b
↦ [2]:
↦    _[1]=3
↦    _[2]=a
```

D.4.27.4 radicalZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Usage: radicalZ(I); I ideal
The coefficients must be ZZ.

Return: the radcal of the input ideal

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2);
I=intersectZ(I,I3);
I=intersectZ(I,I4);
```

```
I=intersectZ(I,I5);
I=intersectZ(I,I6);
radicalZ(I);
↦ _[1]=102bd
↦ _[2]=78ad
↦ _[3]=51bc
↦ _[4]=39ac
↦ _[5]=6abd
↦ _[6]=3abc
ideal J=intersectZ(ideal(17,a),ideal(17,a2,b));
radicalZ(J);
↦ _[1]=17
↦ _[2]=a
```

D.4.27.5 heightZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Usage: heightZ(I); I ideal
The coefficients must be ZZ.

Return: the height of the input ideal

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2);
I=intersectZ(I,I3);
I=intersectZ(I,I4);
I=intersectZ(I,I5);
I=intersectZ(I,I6);
heightZ(I);
↦ 1
```

D.4.27.6 equidimZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Usage: equidimZ(I); I ideal
The coefficients must be ZZ.

Return: the part of minimal height

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2);
I=intersectZ(I,I3);
I=intersectZ(I,I4);
I=intersectZ(I,I5);
I=intersectZ(I,I6);
equidimZ(I);
↦ _[1]=3
```

D.4.27.7 intersectZ

Procedure from library `primdecint.lib` (see Section D.4.27 [primdecint_lib], page 1252).

Return: the intersection of the input ideals

Note: this is an alternative to intersect(I,J) over integers, is faster for some examples and should be kept for debug purposes.

Example:

```
LIB "primdecint.lib";
ring R=integer,(a,b,c,d),dp;
ideal I1=9,a,b;
ideal I2=3,c;
ideal I3=11,2a,7b;
ideal I4=13a2,17b4;
ideal I5=9c5,6d5;
ideal I6=17,a15,b15,c15,d15;
ideal I=intersectZ(I1,I2); I;
↦ I[1]=9
↦ I[2]=3b
↦ I[3]=3a
↦ I[4]=bc
↦ I[5]=ac
I=intersectZ(I,I3); I;
↦ I[1]=99
↦ I[2]=3b
↦ I[3]=3a
↦ I[4]=bc
↦ I[5]=ac
I=intersectZ(I,I4); I;
↦ I[1]=39a2
↦ I[2]=13a2c
↦ I[3]=51b4
↦ I[4]=17b4c
↦ I[5]=3a2b4
↦ I[6]=a2b4c
I=intersectZ(I,I5); I;
↦ I[1]=78a2d5
↦ I[2]=117a2c5
↦ I[3]=102b4d5
↦ I[4]=153b4c5
↦ I[5]=6a2b4d5
↦ I[6]=9a2b4c5
↦ I[7]=39a2c5d5
↦ I[8]=51b4c5d5
↦ I[9]=3a2b4c5d5
I=intersectZ(I,I6); I;
↦ I[1]=1326a2d5
↦ I[2]=1989a2c5
↦ I[3]=102b4d5
↦ I[4]=153b4c5
↦ I[5]=663a2c5d5
↦ I[6]=51b4c5d5
↦ I[7]=78a2d15
↦ I[8]=117a2c15
↦ I[9]=78a15d5
↦ I[10]=117a15c5
↦ I[11]=6a2b4d15
↦ I[12]=9a2b4c15
↦ I[13]=39a2c5d15
↦ I[14]=39a2c15d5
↦ I[15]=6a2b15d5
↦ I[16]=9a2b15c5
↦ I[17]=6a15b4d5
↦ I[18]=9a15b4c5
↦ I[19]=39a15c5d5
↦ I[20]=3a2b4c5d15
↦ I[21]=3a2b4c15d5
↦ I[22]=3a2b15c5d5
↦ I[23]=3a15b4c5d5
```

D.4.28 primitiv_lib

Library: primitiv.lib

Purpose: Computing a Primitive Element

Author: Martin Lamm, email: lamm@mathematik.uni-kl.de

Procedures:

D.4.28.1 primitive

Procedure from library `primitiv.lib` (see Section D.4.28 [primitiv_lib], page 1259).

Usage: primitive(i); i ideal

Assume: i is given by generators m[1],...,m[n] such that for j=1,...,n
- m[j] is a polynomial in k[x(1),...,x(j)]
- m[j](a[1],...,a[j-1],x(j)) is the minimal polynomial for a[j] over k(a[1],...,a[j-1])
(k the ground field of the current basering and x(1),...,x(n) the ring variables).

Return: ideal j in k[x(n)] with
- j[1] a minimal polynomial for a primitive element b of k(a[1],...,a[n]) over k,
- j[2],...,j[n+1] polynomials in k[x(n)] such that j[i+1](b)=a[i] for i=1,...,n.

Note: the number of variables in the basering has to be exactly n, the number of given generators (i.e., minimal polynomials).

If the ground field k has only a few elements it may happen that no linear combination of a[1],...,a[n] is a primitive element. In this case primitive(i) returns the zero ideal, and one should use primitive_extra(i) instead.

Example:

```
LIB "primitiv.lib";
ring exring=0,(x,y),dp;
ideal i=x2+1,y2-x;                  // compute Q(i,i^(1/2))=:L
ideal j=primitive(i);
j[1];                               // L=Q(a) with a=(-1)^(1/4)
↦ y4+1
j[2];                               // i=a^2
↦ y2
j[3];                               // i^(1/2)=a
↦ y
// the 2nd element was already primitive!
j=primitive(ideal(x2-2,y2-3));      // compute Q(sqrt(2),sqrt(3))
j[1];
↦ y4-10y2+1
j[2];
↦ 1/2y3-9/2y
j[3];
↦ -1/2y3+11/2y
// no element was primitive -- the calculation of primitive elements
// is based on a random choice.
```

See also: Section D.4.28.2 [primitive_extra], page 1260.

D.4.28.2 primitive_extra

Procedure from library primitiv.lib (see Section D.4.28 [primitiv_lib], page 1259).

Usage: primitive_extra(i); i ideal

Assume: The ground field of the basering is k=Q or k=Z/pZ and the ideal i is given by 2 generators f,g with the following properties:
f is the minimal polynomial of a in k[x],
g is a polynomial in k[x,y] s.th. g(a,y) is the minpoly of b in k(a)[y].
Here, x is the name of the first ring variable, y the name of the second.

Return: ideal j in k[y] such that
j[1] is the minimal polynomial for a primitive element c of k(a,b) over k,
j[2] is a polynomial s.th. j[2](c)=a.

Note: While primitive(i) may fail for finite fields, primitive_extra(i) tries all elements of k(a,b) and, hence, always finds a primitive element.
In order to do this (try all elements), field extensions like Z/pZ(a) are not allowed for the ground field k.
primitive_extra(i) assumes that the second generator, g, is monic as polynomial in (k[x])[y].

Example:

```
LIB "primitiv.lib";
ring exring=3,(x,y),dp;
ideal i=x2+1,y3+y2-1;
```

```
primitive_extra(i);
↦ _[1]=y6-y5+y4-y3-y-1
↦ _[2]=y5+y4+y2+y+1
ring extension=(3,y),x,dp;
minpoly=y6-y5+y4-y3-y-1;
number a=y5+y4+y2+y+1;
a^2;
↦ -1
factorize(x2+1);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+(y5+y4+y2+y+1)
↦    _[3]=x+(-y5-y4-y2-y-1)
↦ [2]:
↦    1,1,1
factorize(x3+x2-1);
↦ [1]:
↦    _[1]=1
↦    _[2]=x+(y3+y+1)
↦    _[3]=x+(y5+y4+y2+1)
↦    _[4]=x+(-y5-y4-y3-y2-y-1)
↦ [2]:
↦    1,1,1,1
```

D.4.28.3 splitring

Procedure from library `primitiv.lib` (see Section D.4.28 [primitiv_lib], page 1259).

Usage: splitring(f[,L]); f poly, L list of polys and/or ideals (optional)

Assume: f is univariate and irreducible over the active ring.
The active ring must allow an algebraic extension (e.g., it cannot be a transcendent ring extension of Q or Z/p).

Return: ring;
if called with a nonempty second parameter L, then in the output ring there is defined a list erg (=L mapped to the new ring); if the minpoly of the active ring is non-zero, then the image of the primitive root of f in the output ring is appended as last entry of the list erg.

Note: If the old ring has no parameter, the name `a` is chosen for the parameter of R (if `a` is no ring variable; if it is, `b` is chosen, etc.; if `a,b,c,o` are ring variables, `splitring(f[,L])` produces an error message), otherwise the name of the parameter is kept and only the minimal polynomial is changed.
The names of the ring variables and the orderings are not affected.

Example:

```
LIB "primitiv.lib";
ring r=0,(x,y),dp;
def r1=splitring(x2-2);
setring r1; basering;    // change to Q(sqrt(2))
↦ // coefficients: QQ[a]/(a2-2)
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                    : names    x y
↦ //        block   2 : ordering C
// change to Q(sqrt(2),sqrt(sqrt(2)))=Q(a) and return the transformed
// old parameter:
def r2=splitring(x2-a,a);
↦ // new minimal polynomial: a4-2
setring r2; basering; erg;
↦ // coefficients: QQ[a]/(a4-2)
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                    : names    x y
↦ //        block   2 : ordering C
↦ [1]:
↦    (a2)
↦ [2]:
↦    (a)
// the result is (a)^2 = (sqrt(sqrt(2)))^2
kill r1; kill r2;
```

D.4.29 realrad_lib

Library: realrad.lib

Purpose: Computation of real radicals

Author : Silke Spang

Overview: Algorithms about the computation of the real
radical of an arbitrary ideal over the rational numbers and transcendetal extensions thereof

Procedures:

D.4.29.1 realpoly

Procedure from library `realrad.lib` (see Section D.4.29 [realrad_lib], page 1262).

Usage: realpoly(f); a univariate polynomial f;

Return: poly f, where f is the real part of the input f

Example:

```
LIB "realrad.lib";
ring r1 = 0,x,dp;
poly f=x5+16x2+x+1;
realpoly(f);
↦ x5+16x2+x+1
realpoly(f*(x4+2));
↦ x5+16x2+x+1
ring r2=0,(x,y),dp;
poly f=x6-3x4y2 + y6 + x2y2 -6y+5;
realpoly(f);
↦ x6-3x4y2+y6+x2y2-6y+5
ring r3=0,(x,y,z),dp;
poly f=x4y4-2x5y3z2+x6y2z4+2x2y3z-4x3y2z3+2x4yz5+z2y2-2z4yx+z6x2;
realpoly(f);
```

```
↦ x3yz2-x2y2+xz3-yz
realpoly(f*(x2+y2+1));
↦ x3yz2-x2y2+xz3-yz
```

D.4.29.2 realzero

Procedure from library `realrad.lib` (see Section D.4.29 [realrad_lib], page 1262).

Usage: realzero(j); a zero-dimensional ideal j

Return: j: a zero dimensional ideal, which is the real radical of i, if dim(i)=0
0: otherwise
this acts via
primary decomposition (i=1)
listdecomp (i=2) or facstd (i=3)

Example:

```
LIB "realrad.lib";
//in non parametric fields
ring r=0,(x,y),dp;
ideal i=(y3+3y2+y+1)*(y2+4y+4)*(x2+1),(x2+y)*(x2-y2)*(x2+2xy+y2)*(y2+y+1);
realzero(i);
↦ _[1]=y4+5y3+7y2+3y+2
↦ _[2]=x4-x2y2+x2y-y3
ideal j=(y3+3y2+y+1)*(y2-2y+1),(x2+y)*(x2-y2);
realzero(j);
↦ _[1]=y4+2y3-2y2-1
↦ _[2]=x2y3+3x2y2+x2y-y3+x2-3y2-y-1
↦ _[3]=x4-x2y2+x2y-y3
//to get every path
ring r1=(0,t),(x,y),lp;
ideal m1=x2+1-t,y3+t2;
ideal m2=x2+t2+1,y2+t;
ideal m3=x2+1-t,y2-t;
ideal m4=x^2+1+t,y2-t;
ideal i=intersect(m1,m2,m3,m4);
realzero(i);
↦ _[1]=y5+(-t)*y3+(t2)*y2+(-t3)
↦ _[2]=x2+(-t+1)
```

D.4.29.3 realrad

Procedure from library `realrad.lib` (see Section D.4.29 [realrad_lib], page 1262).

Usage: realrad(id), id an ideal of arbitrary dimension

Return: the real radical of id

Exampe: example realrad; shows an example

Example:

```
LIB "realrad.lib";
ring r1=0,(x,y,z),lp;
//dimension 0
ideal i0=(x2+1)*(x3-2),(y3-2)*(y2+y+1),z3+2;
//dimension 1
```

```
ideal i1=(y3+3y2+y+1)*(y2+4y+4)*(x2+1),(x2+y)*(x2-y2)*(x2+2xy+y2)*(y2+y+1);
ideal i=intersect(i0,i1);
realrad(i);
↦ _[1]=2x4y+5x4-2x2y3-3x2y2+5x2y-2y4-5y3
↦ _[2]=-3x4+4x2y4+20x2y3+31x2y2+9x2y+8x2+4y5-4y4z2+4y4z+20y4-20y3z2+20y3z+3\
   1y3-28y2z2+28y2z+12y2-12yz2+12yz+8y-8z2+8z
↦ _[3]=y4z3+2y4+5y3z3+10y3+7y2z3+14y2+3yz3+6y+2z3+4
↦ _[4]=59y5z2+38y5z-74y5+333y4z2+116y4z-488y4+603y3z2-104y3z-1108y3+443y2z2\
   -404y2z-1048y2+232yz2-146yz-502y+76z2-148z-236
↦ _[5]=59xy4z2+38xy4z-74xy4+295xy3z2+190xy3z-370xy3+413xy2z2+266xy2z-518xy2\
   +177xyz2+114xyz-222xy+118xz2+76xz-148x+38y4z2-74y4z-118y4+190y3z2-370y3z-\
   590y3+266y2z2-518y2z-826y2+114yz2-222yz-354y+76z2-148z-236
```

D.4.30 reesclos_lib

Library: reesclos.lib

Purpose: procedures to compute the int. closure of an ideal

Author: Tobias Hirsch, email: hirsch@math.tu-cottbus.de
Janko Boehm, email: boehm@mathematik.uni-kl.de
Magdaleen Marais, email: magdaleen@aims.ac.za

Overview: A library to compute the integral closure of an ideal I in a polynomial ring R=k[x(1),...,x(n)] using the Rees Algebra R[It] of I. It computes the integral closure of R[It],
which is a graded subalgebra of R[t]. The degree-k-component is the integral closure of the k-th power of I.

In contrast to the previous version, the library uses 'normal.lib' to compute the integral closure of R[It]. This improves the performance considerably.

Procedures:

D.4.30.1 ReesAlgebra

Procedure from library `reesclos.lib` (see Section D.4.30 [reesclos_lib], page 1264).

Usage: ReesAlgebra (I); I = ideal

Return: The Rees algebra R[It] as an affine ring, where I is an ideal in R. The procedure returns a list containing two rings:
[1]: a ring, say RR; in the ring an ideal ker such that R[It]=RR/ker

[2]: a ring, say Kxt; the basering with additional variable t containing an ideal mapI that defines the map RR–>Kxt

Example:

```
LIB "reesclos.lib";
ring R = 0,(x,y),dp;
ideal I = x2,xy4,y5;
list L = ReesAlgebra(I);
def Rees = L[1];       // defines the ring Rees, containing the ideal ker
setring Rees;          // passes to the ring Rees
Rees;
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y U(1) U(2) U(3)
↦ //        block   2 : ordering C
ker;                  // R[It] is isomorphic to Rees/ker
↦ ker[1]=y*U(2)-x*U(3)
↦ ker[2]=y^3*U(1)*U(3)-U(2)^2
↦ ker[3]=y^4*U(1)-x*U(2)
↦ ker[4]=x*y^2*U(1)*U(3)^2-U(2)^3
↦ ker[5]=x^2*y*U(1)*U(3)^3-U(2)^4
↦ ker[6]=x^3*U(1)*U(3)^4-U(2)^5
```

D.4.30.2 normalI

Procedure from library `reesclos.lib` (see Section D.4.30 [reesclos_lib], page 1264).

Usage: normalI (I [,p [,r [,l]]]); I an ideal, p, r, and l optional integers

Return: the integral closure of I, ..., I^p, where I is an ideal in the polynomial ring R=k[x(1),...x(n)]. If p is not given, or p==0, compute the closure of all powers up to the maximum degree in t occurring in the closure of R[It] (so this is the last power whose closure is not just the sum/product of the smaller). If r is given and r==1, normalI starts with a check whether I is already a radical ideal.
If l==1 then locNormal instead of normal is used to compute normalization. The result is a list containing the closure of the desired powers of I as ideals of the basering.

Display: The procedure displays more comments for higher printlevel.

Example:

```
LIB "reesclos.lib";
ring R=0,(x,y),dp;
ideal I = x2,xy4,y5;
list J = normalI(I);
I;
↦ I[1]=x2
↦ I[2]=xy4
↦ I[3]=y5
J;                              // J[1] is the integral closure of I
↦ [1]:
↦    _[1]=x2
↦    _[2]=xy4
↦    _[3]=y5
↦    _[4]=xy3
```

D.4.31 rstandard_lib

Library: rstandard.lib

Purpose: Computes Janet bases and border bases for ideals

Authors: Shamsa Kanwal lotus_zone16@yahoo.com
Gerhard Pfister pfister@mathematik.uni-kl.de

Overview: Computing Janet bases and border bases for any ordering using the idea of r-standard bases (defined by V. Gerdt)

References:

[1] A. Kehrein, M. Kreuzer, L. Robbiano: An algebrists view on border bases, in: A. Dickenstein and I. Emiris (eds.), Solving Polynomial Equations: Foundations, Algorithms and Applications, Springer, Heidelberg 2005, 169-202.

[2] V.P. Gerdt: Involute Algorithms for Computing Groebner Bases, In Computational Commutative and Non-Computational Algebra Geometry, S.Conjocaru, G. Pfister and V. Ufnarovski (Eds.), NATO Science Series,105 Press 2005, 199-255.

Procedures:

D.4.31.1 borderBasis

Procedure from library `rstandard.lib` (see Section D.4.31 [rstandard_lib], page 1265).

Usage: borderBasis(I); I is an ideal.

Return: ideal, a border basis for I.

Purpose: Computes a border basis for the ideal given by the generators in I.

Example:

```
LIB "rstandard.lib";
ring R=32003,(x,y,z),ds;
poly f=x3y+x5+x3y2+2x2y3+x2yz2+xy5+x12+y16+z20;
ideal i= jacob(f);
i=i,f;
ideal j=borderBasis(i);  j;
↦ j[1]=y4z20
↦ j[2]=y5z19
↦ j[3]=xy4z19
↦ j[4]=y6z18
↦ j[5]=xy5z18
↦ j[6]=y3z20
↦ j[7]=xy3z19
↦ j[8]=xy4z18
↦ j[9]=y6z17-12804y4z19
↦ j[10]=xy5z17
↦ j[11]=y7z16+16001y5z18
↦ j[12]=xy6z16
↦ j[13]=y8z15-6401y4z19
↦ j[14]=y9z14
↦ j[15]=xy8z14
↦ j[16]=z22
↦ j[17]=yz21-2461y4z19
↦ j[18]=xz21
↦ j[19]=y2z20
↦ j[20]=xyz20
↦ j[21]=xy2z19+y4z19
↦ j[22]=xy3z18+8001y5z18
↦ j[23]=xy4z17-12801y4z19
↦ j[24]=xy5z16
↦ j[25]=y7z15+16001y5z17+y3z19+6153y4z19
↦ j[26]=xy6z15
↦ j[27]=xy7z14
↦ j[28]=y9z13
↦ j[29]=xy8z13
↦ j[30]=xz20
↦ j[31]=xyz19+y3z19+6153y4z19
↦ j[32]=xy2z18-y8z14-16001y6z16
↦ j[33]=xy3z17+8001y5z17+16001y3z19+12925y4z19
↦ j[34]=xy4z16-16001y8z14
↦ j[35]=xy5z15
↦ j[36]=xy6z14
↦ j[37]=xy7z13
↦ j[38]=y9z12
↦ j[39]=xy8z12
↦ j[40]=xz19+y2z19+13336z21-10668y3z19-2051y4z19
↦ j[41]=xyz18-y7z14-16001y5z16
↦ j[42]=x2z18+15999y8z14
↦ j[43]=xy2z17-y8z13-16001y6z15
↦ j[44]=xy3z16-16001y7z14
↦ j[45]=xy4z15-16001y8z13
↦ j[46]=xy5z14
↦ j[47]=xy6z13
↦ j[48]=xy7z12
↦ j[49]=y9z11
↦ j[50]=xy8z11
↦ j[51]=xyz17-y7z13-16001y5z15
↦ j[52]=x2z17+15999y8z13
↦ j[53]=xy2z16-y8z12-16001y6z14
↦ j[54]=xy3z15-16001y7z13
↦ j[55]=xy4z14-16001y8z12
↦ j[56]=xy5z13
↦ j[57]=xy6z12
↦ j[58]=xy7z11
↦ j[59]=y9z10
↦ j[60]=xy8z10
↦ j[61]=xyz16-y7z12-16001y5z14
↦ j[62]=x2z16+15999y8z12
↦ j[63]=xy2z15-y8z11-16001y6z13
↦ j[64]=xy3z14-16001y7z12
↦ j[65]=xy4z13-16001y8z11
↦ j[66]=xy5z12
↦ j[67]=xy6z11
↦ j[68]=xy7z10
↦ j[69]=y9z9
↦ j[70]=xy8z9
↦ j[71]=xyz15-y7z11-16001y5z13
↦ j[72]=x2z15+15999y8z11
↦ j[73]=xy2z14-y8z10-16001y6z12
↦ j[74]=xy3z13-16001y7z11
↦ j[75]=xy4z12-16001y8z10
↦ j[76]=xy5z11
↦ j[77]=xy6z10
↦ j[78]=xy7z9
↦ j[79]=y9z8
↦ j[80]=xy8z8
↦ j[81]=xyz14-y7z10-16001y5z12
↦ j[82]=x2z14+15999y8z10
↦ j[83]=xy2z13-y8z9-16001y6z11
↦ j[84]=xy3z12-16001y7z10
↦ j[85]=xy4z11-16001y8z9
↦ j[86]=xy5z10
↦ j[87]=xy6z9
↦ j[88]=xy7z8
↦ j[89]=y9z7
↦ j[90]=xy8z7
↦ j[91]=xyz13-y7z9-16001y5z11
↦ j[92]=x2z13+15999y8z9
↦ j[93]=xy2z12-y8z8-16001y6z10
↦ j[94]=xy3z11-16001y7z9
↦ j[95]=xy4z10-16001y8z8
↦ j[96]=xy5z9
↦ j[97]=xy6z8
↦ j[98]=xy7z7
↦ j[99]=y9z6
↦ j[100]=xy8z6
↦ j[101]=xyz12-y7z8-16001y5z10
↦ j[102]=x2z12+15999y8z8
↦ j[103]=xy2z11-y8z7-16001y6z9
↦ j[104]=xy3z10-16001y7z8
↦ j[105]=xy4z9-16001y8z7
↦ j[106]=xy5z8
↦ j[107]=xy6z7
↦ j[108]=xy7z6
↦ j[109]=y9z5
↦ j[110]=xy8z5
↦ j[111]=xyz11-y7z7-16001y5z9
↦ j[112]=x2z11+15999y8z7
↦ j[113]=xy2z10-y8z6-16001y6z8
↦ j[114]=xy3z9-16001y7z7
↦ j[115]=xy4z8-16001y8z6
↦ j[116]=xy5z7
↦ j[117]=xy6z6
↦ j[118]=xy7z5
↦ j[119]=y9z4
↦ j[120]=xy8z4
↦ j[121]=xyz10-y7z6-16001y5z8
↦ j[122]=x2z10+15999y8z6
↦ j[123]=xy2z9-y8z5-16001y6z7
↦ j[124]=xy3z8-16001y7z6
↦ j[125]=xy4z7-16001y8z5
↦ j[126]=xy5z6
↦ j[127]=xy6z5
↦ j[128]=xy7z4
↦ j[129]=y9z3
↦ j[130]=xy8z3
↦ j[131]=xyz9-y7z5-16001y5z7
↦ j[132]=x2z9+15999y8z5
↦ j[133]=xy2z8-y8z4-16001y6z6
↦ j[134]=xy3z7-16001y7z5
↦ j[135]=xy4z6-16001y8z4
↦ j[136]=xy5z5
↦ j[137]=xy6z4
↦ j[138]=xy7z3
↦ j[139]=y9z2
↦ j[140]=xy8z2
↦ j[141]=xyz8-y7z4-16001y5z6
↦ j[142]=x2z8+15999y8z4
↦ j[143]=xy2z7-y8z3-16001y6z5
↦ j[144]=xy3z6-16001y7z4
↦ j[145]=xy4z5-16001y8z3
↦ j[146]=xy5z4
↦ j[147]=xy6z3
↦ j[148]=xy7z2
↦ j[149]=y9z+10y4z19
↦ j[150]=xy8z
↦ j[151]=xyz7-y7z3-16001y5z5
↦ j[152]=x2z7+15999y8z3-10672z21-10671y3z19+11493y4z19
↦ j[153]=xy2z6-y8z2-16001y6z4
↦ j[154]=x2yz6
↦ j[155]=x3z6
↦ j[156]=xy3z5-16001y7z3
↦ j[157]=xy4z4-16001y8z2
↦ j[158]=xy5z3
↦ j[159]=xy6z2
↦ j[160]=xy7z-10y4z19
↦ j[161]=y9-13yz20
↦ j[162]=xy8
↦ j[163]=xyz6-y7z2-16001y5z4
↦ j[164]=xy2z5-y8z-16001y6z3-10y3z19-7333y4z19
↦ j[165]=x2yz5
↦ j[166]=x3z5+10672z21+10671y3z19-11493y4z19
↦ j[167]=xy3z4-16001y7z2
↦ j[168]=xy4z3-16001y8z+5y3z19-14791y4z19
↦ j[169]=xy5z2
↦ j[170]=xy6z-10y3z19+7388y4z19
↦ j[171]=xy7+13yz20
↦ j[172]=xyz5-y7z-16001y5z3-10y2z19+5333z21-10616y3z19+391y4z19
↦ j[173]=xy2z4-y8-16001y6z2-3x2z6-15994y8z2+13z20+15952yz20
↦ j[174]=x2yz4
↦ j[175]=x3z4+x2z6+15999y8z2
↦ j[176]=xy3z3-16001y7z+5y2z19+10666z21+10646y3z19-9017y4z19
↦ j[177]=x2y2z3-7393y4z19
↦ j[178]=xy4z2-16001y8-16000x2z6+7997y8z2+15995z20+8018yz20
↦ j[179]=xy5z-10y2z19-5333z21+10671y3z19-11493y4z19
↦ j[180]=xy6-3x2z6-15994y8z2+13z20+15yz20
↦ j[181]=x2yz3+10z21
↦ j[182]=x3z3+x2z5+15999y8z-25y3z19-9699y4z19
↦ j[183]=xy3z2-16001xyz4+8001y5z2+8002x2z6+11998y8z2+15994z20+15994yz20
↦ j[184]=x2y2z2+10yz20
↦ j[185]=xy4z-16001xy2z3+8001y6z+15994yz19+15994y2z19+15989y4z19
↦ j[186]=xy5-xyz4+y7+16001y5z2+15987x2z6+8037y8z2-15937z20-15931yz20
↦ j[187]=x2y4+3x2z6+15994y8z2-13z20+15983yz20
↦ j[188]=x2yz2+10z20
↦ j[189]=x3z2+x2z4+15999y8+15994x2z6-7982y8z2-15969z20-8147yz20
↦ j[190]=xy3z-16001xyz3+8001y5z+8002x2z5+11998y8z+15994z19+15994yz19+7957y3\
   z19+14472y4z19
↦ j[191]=x2y2z+10yz19
↦ j[192]=x2y3+xyz4-y7-16001y5z2-15986x2z6+7962y8z2-68z20+15931yz20
↦ j[193]=x2yz+10z19
↦ j[194]=x3z+x2z3+15999xy2z3-8002y6z+15979yz19-15944y2z19+10696z21+10661y3z\
   19+7047y4z19
↦ j[195]=x2y2+10669xy4-10667xy2z2+10668y6-xyz4+y7+16001y5z2+15986x2z6-7962y\
   8z2+68z20+5426yz20
↦ j[196]=x2y+10669xy3-10667xyz2-10669xy4+10668y5+10667xy2z2-10668y6-10666x2\
   z4+xyz4-y7-16001y5z2-5338y8-2x2z6+5y8z2-5326z20-13868yz20
↦ j[197]=x3+x2z2-3xy4-4xy2z2-2y6+8xyz4-8y7+4y5z2+122x2z6-305y8z2-507z20-694\
   yz20
```

See also: Section D.4.31.2 [modBorder], page 1270.

D.4.31.2 modBorder

Procedure from library `rstandard.lib` (see Section D.4.31 [rstandard_lib], page 1265).

Usage: modBorder(I,i); I is an ideal, i an integer.

Return: ideal, a border basis for I using modular methods.

Purpose: Computes a border basis for the ideal given by the generators in I using modular techniques.
If second argument is 0 then the result is not verified.

Example:

```
LIB "rstandard.lib";
ring R=0,(x,y,z),ds;
poly f=x3y+x5+x3y2+2x2y3+x2yz2+xy5+x12+y16+z20;
ideal i= jacob(f);
i=i,f;
ideal j=modBorder(i,1);  j;
↦ j[1]=y4z20
↦ j[2]=y5z19
↦ j[3]=xy4z19
↦ j[4]=y6z18
↦ j[5]=xy5z18
↦ j[6]=y3z20
↦ j[7]=xy3z19
↦ j[8]=xy4z18
↦ j[9]=y6z17-14/5y4z19
↦ j[10]=xy5z17
↦ j[11]=y7z16-1/2y5z18
↦ j[12]=xy6z16
↦ j[13]=y8z15-2/5y4z19
↦ j[14]=y9z14
↦ j[15]=xy8z14
↦ j[16]=z22
↦ j[17]=yz21+10/13y4z19
↦ j[18]=xz21
↦ j[19]=y2z20
↦ j[20]=xyz20
↦ j[21]=xy2z19+y4z19
↦ j[22]=xy3z18+1/4y5z18
↦ j[23]=xy4z17+1/5y4z19
↦ j[24]=xy5z16
↦ j[25]=y7z15-1/2y5z17+y3z19-37/26y4z19
↦ j[26]=xy6z15
↦ j[27]=xy7z14
↦ j[28]=y9z13
↦ j[29]=xy8z13
↦ j[30]=xz20
↦ j[31]=xyz19+y3z19-37/26y4z19
↦ j[32]=xy2z18-y8z14+1/2y6z16
↦ j[33]=xy3z17+1/4y5z17-1/2y3z19+37/52y4z19
↦ j[34]=xy4z16+1/2y8z14
↦ j[35]=xy5z15
↦ j[36]=xy6z14
↦ j[37]=xy7z13
↦ j[38]=y9z12
↦ j[39]=xy8z12
↦ j[40]=xz19+y2z19+17/12z21-1/3y3z19+37/78y4z19
↦ j[41]=xyz18-y7z14+1/2y5z16
↦ j[42]=x2z18-5/2y8z14
↦ j[43]=xy2z17-y8z13+1/2y6z15
↦ j[44]=xy3z16+1/2y7z14
↦ j[45]=xy4z15+1/2y8z13
↦ j[46]=xy5z14
↦ j[47]=xy6z13
↦ j[48]=xy7z12
↦ j[49]=y9z11
↦ j[50]=xy8z11
↦ j[51]=xyz17-y7z13+1/2y5z15
↦ j[52]=x2z17-5/2y8z13
↦ j[53]=xy2z16-y8z12+1/2y6z14
↦ j[54]=xy3z15+1/2y7z13
↦ j[55]=xy4z14+1/2y8z12
↦ j[56]=xy5z13
↦ j[57]=xy6z12
↦ j[58]=xy7z11
↦ j[59]=y9z10
↦ j[60]=xy8z10
↦ j[61]=xyz16-y7z12+1/2y5z14
↦ j[62]=x2z16-5/2y8z12
↦ j[63]=xy2z15-y8z11+1/2y6z13
↦ j[64]=xy3z14+1/2y7z12
↦ j[65]=xy4z13+1/2y8z11
↦ j[66]=xy5z12
↦ j[67]=xy6z11
↦ j[68]=xy7z10
↦ j[69]=y9z9
↦ j[70]=xy8z9
↦ j[71]=xyz15-y7z11+1/2y5z13
↦ j[72]=x2z15-5/2y8z11
↦ j[73]=xy2z14-y8z10+1/2y6z12
↦ j[74]=xy3z13+1/2y7z11
↦ j[75]=xy4z12+1/2y8z10
↦ j[76]=xy5z11
↦ j[77]=xy6z10
↦ j[78]=xy7z9
↦ j[79]=y9z8
↦ j[80]=xy8z8
↦ j[81]=xyz14-y7z10+1/2y5z12
↦ j[82]=x2z14-5/2y8z10
↦ j[83]=xy2z13-y8z9+1/2y6z11
↦ j[84]=xy3z12+1/2y7z10
↦ j[85]=xy4z11+1/2y8z9
↦ j[86]=xy5z10
↦ j[87]=xy6z9
↦ j[88]=xy7z8
↦ j[89]=y9z7
↦ j[90]=xy8z7
↦ j[91]=xyz13-y7z9+1/2y5z11
↦ j[92]=x2z13-5/2y8z9
↦ j[93]=xy2z12-y8z8+1/2y6z10
↦ j[94]=xy3z11+1/2y7z9
↦ j[95]=xy4z10+1/2y8z8
↦ j[96]=xy5z9
↦ j[97]=xy6z8
↦ j[98]=xy7z7
↦ j[99]=y9z6
↦ j[100]=xy8z6
↦ j[101]=xyz12-y7z8+1/2y5z10
↦ j[102]=x2z12-5/2y8z8
↦ j[103]=xy2z11-y8z7+1/2y6z9
↦ j[104]=xy3z10+1/2y7z8
↦ j[105]=xy4z9+1/2y8z7
↦ j[106]=xy5z8
↦ j[107]=xy6z7
↦ j[108]=xy7z6
↦ j[109]=y9z5
↦ j[110]=xy8z5
↦ j[111]=xyz11-y7z7+1/2y5z9
↦ j[112]=x2z11-5/2y8z7
↦ j[113]=xy2z10-y8z6+1/2y6z8
↦ j[114]=xy3z9+1/2y7z7
↦ j[115]=xy4z8+1/2y8z6
↦ j[116]=xy5z7
↦ j[117]=xy6z6
↦ j[118]=xy7z5
↦ j[119]=y9z4
↦ j[120]=xy8z4
↦ j[121]=xyz10-y7z6+1/2y5z8
↦ j[122]=x2z10-5/2y8z6
↦ j[123]=xy2z9-y8z5+1/2y6z7
↦ j[124]=xy3z8+1/2y7z6
↦ j[125]=xy4z7+1/2y8z5
↦ j[126]=xy5z6
↦ j[127]=xy6z5
↦ j[128]=xy7z4
↦ j[129]=y9z3
↦ j[130]=xy8z3
↦ j[131]=xyz9-y7z5+1/2y5z7
↦ j[132]=x2z9-5/2y8z5
↦ j[133]=xy2z8-y8z4+1/2y6z6
↦ j[134]=xy3z7+1/2y7z5
↦ j[135]=xy4z6+1/2y8z4
↦ j[136]=xy5z5
↦ j[137]=xy6z4
↦ j[138]=xy7z3
↦ j[139]=y9z2
↦ j[140]=xy8z2
↦ j[141]=xyz8-y7z4+1/2y5z6
↦ j[142]=x2z8-5/2y8z4
↦ j[143]=xy2z7-y8z3+1/2y6z5
↦ j[144]=xy3z6+1/2y7z4
↦ j[145]=xy4z5+1/2y8z3
↦ j[146]=xy5z4
↦ j[147]=xy6z3
↦ j[148]=xy7z2
↦ j[149]=y9z+10y4z19
↦ j[150]=xy8z
↦ j[151]=xyz7-y7z3+1/2y5z5
↦ j[152]=x2z7-5/2y8z3-13/3z21-10/3y3z19+185/39y4z19
↦ j[153]=xy2z6-y8z2+1/2y6z4
↦ j[154]=x2yz6
↦ j[155]=x3z6
↦ j[156]=xy3z5+1/2y7z3
↦ j[157]=xy4z4+1/2y8z2
↦ j[158]=xy5z3
↦ j[159]=xy6z2
↦ j[160]=xy7z-10y4z19
↦ j[161]=y9-13yz20
↦ j[162]=xy8
↦ j[163]=xyz6-y7z2+1/2y5z4
↦ j[164]=xy2z5-y8z+1/2y6z3-10y3z19+680/13y4z19
↦ j[165]=x2yz5
↦ j[166]=x3z5+13/3z21+10/3y3z19-185/39y4z19
↦ j[167]=xy3z4+1/2y7z2
↦ j[168]=xy4z3+1/2y8z+5y3z19-265/13y4z19
↦ j[169]=xy5z2
↦ j[170]=xy6z-10y3z19+35/13y4z19
↦ j[171]=xy7+13yz20
↦ j[172]=xyz5-y7z+1/2y5z3-10y2z19-5/6z21+155/3y3z19-1505/78y4z19
↦ j[173]=xy2z4-y8+1/2y6z2-3x2z6+15/2y8z2+13z20-99/2yz20
↦ j[174]=x2yz4
↦ j[175]=x3z4+x2z6-5/2y8z2
↦ j[176]=xy3z3+1/2y7z+5y2z19-5/3z21-65/3y3z19+370/39y4z19
↦ j[177]=x2y2z3-100/13y4z19
↦ j[178]=xy4z2+1/2y8+3/2x2z6-15/4y8z2-13/2z20+69/4yz20
↦ j[179]=xy5z-10y2z19+5/6z21+10/3y3z19-185/39y4z19
↦ j[180]=xy6-3x2z6+15/2y8z2+13z20+15yz20
↦ j[181]=x2yz3+10z21
↦ j[182]=x3z3+x2z5-5/2y8z-25y3z19+1925/13y4z19
↦ j[183]=xy3z2+1/2xyz4+1/4y5z2+5/4x2z6-25/8y8z2-15/2z20-15/2yz20
↦ j[184]=x2y2z2+10yz20
↦ j[185]=xy4z+1/2xy2z3+1/4y6z-15/2yz19-15/2y2z19-25/2y4z19
↦ j[186]=xy5-xyz4+y7-1/2y5z2-29/2x2z6+145/4y8z2+129/2z20+141/2yz20
↦ j[187]=x2y4+3x2z6-15/2y8z2-13z20-37/2yz20
↦ j[188]=x2yz2+10z20
↦ j[189]=x3z2+x2z4-5/2y8-15/2x2z6+75/4y8z2+65/2z20-585/4yz20
↦ j[190]=xy3z+1/2xyz3+1/4y5z+5/4x2z5-25/8y8z-15/2z19-15/2yz19-175/4y3z19+16\
   475/52y4z19
↦ j[191]=x2y2z+10yz19
↦ j[192]=x2y3+xyz4-y7+1/2y5z2+31/2x2z6-155/4y8z2-68z20-141/2yz20
↦ j[193]=x2yz+10z19
↦ j[194]=x3z+x2z3-5/2xy2z3-5/4y6z-45/2yz19+115/2y2z19+85/3z21-20/3y3z19+561\
   5/78y4z19
↦ j[195]=x2y2+4/3xy4+2/3xy2z2+1/3y6-xyz4+y7-1/2y5z2-31/2x2z6+155/4y8z2+68z2\
   0+553/6yz20
↦ j[196]=x2y+4/3xy3+2/3xyz2-4/3xy4+1/3y5-2/3xy2z2-1/3y6+5/3x2z4+xyz4-y7+1/2\
   y5z2-25/6y8-2x2z6+5y8z2+47/6z20-6401/12yz20
↦ j[197]=x3+x2z2-3xy4-4xy2z2-2y6+8xyz4-8y7+4y5z2+122x2z6-305y8z2-507z20-694\
   yz20
ring S=0,(x,y,z),ds;
ideal i=
3x2+6xy+3y2+6xz+6yz+3z2+3x2yz+4xy2z+y3z+4xyz2+2y2z2+yz3+10x9,
3x2+6xy+3y2+6xz+6yz+3z2+x3z+4x2yz+3xy2z+2x2z2+4xyz2+xz3+10y9,
3x2+6xy+3y2+6xz+6yz+3z2+x3y+2x2y2+xy3+4x2yz+4xy2z+3xyz2+10z9;
ideal j=modBorder(i,0);  j;
↦ j[1]=z25
↦ j[2]=yz24
↦ j[3]=xz24
↦ j[4]=yz23+1/2z24
↦ j[5]=xz23+1/2z24
↦ j[6]=y2z22+1/2z24
↦ j[7]=xyz22-z24
↦ j[8]=xz22+yz22+z23
↦ j[9]=y2z21+yz22+z23
↦ j[10]=xyz21-z23
↦ j[11]=xz21+yz21+z22
↦ j[12]=y2z20+yz21+z22
↦ j[13]=xyz20-z22
↦ j[14]=xz20+yz20+z21
↦ j[15]=y2z19+yz20+z21
↦ j[16]=xyz19-z21
↦ j[17]=xz19+yz19+z20
↦ j[18]=y2z18+yz19+z20
↦ j[19]=xyz18-z20
↦ j[20]=xz18+yz18+z19+5z24
↦ j[21]=y2z17+yz18+z19-22805/228z24
↦ j[22]=xyz17-z19+22235/228z24
↦ j[23]=xz17+yz17+z18+5z23
↦ j[24]=y2z16+yz17+z18+5yz22-22235/228z23+3371625/32z24
↦ j[25]=xyz16-z18+22235/228z23-3371625/32z24
↦ j[26]=y3z15-z18+13985/228yz22+40345/228z23-10114875/64z24
↦ j[27]=xy2z15-yz17-16265/228yz22-9625/114z23+3371625/64z24
↦ j[28]=xz16+yz16+z17+5z22
↦ j[29]=xyz15+y2z15+yz16+5yz21
↦ j[30]=y3z14+3/2y2z15+3/2yz16+1/2z17+15695/228yz21+13985/456z22-70875/32yz\
  22-70875/64z23
↦ j[31]=xy2z14-1/2y2z15-3/2yz16-1/2z17-16835/228yz21-16265/456z22+70875/32y\
  z22+70875/64z23
↦ j[32]=y4z13-39/35y2z15-74/35yz16-39/35z17-114311/798yz21-126587/1596z22+7\
  0875/16yz22-7729425/224z23+14531794875/896z24
↦ j[33]=xy3z13-27/70y2z15+43/70yz16+43/70z17+118757/1596yz21+171239/3192z22\
  -70875/32yz22+15954975/448z23-14531794875/896z24
↦ j[34]=xz15+yz15+z16+5z21-151575/8z24
↦ j[35]=xyz14+y2z14+yz15+5yz20+151575/16z24
↦ j[36]=xy2z13+y3z13+y2z14-5yz20-5z21-111675/16z24
↦ j[37]=y4z12+2y3z13+66/35y2z14+31/35yz15-4/35z16-512487/2156y2z15-512487/2\
  156yz16-512487/2156z17-39/7yz20-2391/133z21-2562435/2156yz21-58384395/431\
  2z22-4757545125/539z23+913078227343185/327712z24
↦ j[38]=xy3z12-y3z13-66/35y2z14-31/35yz15+4/35z16+512487/2156y2z15+512487/2\
  156yz16+512487/2156z17+39/7yz20+3056/133z21+2562435/2156yz21+58384395/431\
  2z22+4757545125/539z23-913076348631735/327712z24
↦ j[39]=y5z11-30/17y3z13-281/119y2z14-162/119yz15+48/119z16+2562435/4312y2z\
  15+2562435/4312yz16+2562435/4312z17+8140/357yz20+354395/6783z21+823087575\
  /146608yz21+10312820775/293216z22-39639375/2176yz22+51740725336875/234572\
  8z23-77611672933260075/11142208z24
↦ j[40]=xy4z11-4/17y3z13+283/595y2z14+283/595yz15-172/595z16-1537461/4312y2\
  z15-1537461/4312yz16-1537461/4312z17-4366/357yz20-232454/6783z21-64884199\
  5/146608yz21-6342681915/293216z22+39639375/2176yz22-31035888952875/234572\
  8z23+46566963638175885/11142208z24
↦ j[41]=y6z10+5/17y3z13+78/119y2z14+78/119yz15-76/119z16-3144177/4312y2z15-\
  3144177/4312yz16-3144177/4312z17-4825/119yz20-183885/2261z21-1696930965/1\
  46608yz21-13353125805/293216z22+118918125/2176yz22-65428421864625/2345728\
  z23+19993596022976407 5/22284416z24
↦ j[42]=y7z9+232/187y3z13+348/187y2z14+161/187yz15+116/187z16+41553/88y2z15\
  +41553/88yz16+41553/88z17+7785/187yz20+290710/3553z21+42390135/2992yz21+1\
  97137395/5984z22-2972953125/23936yz22+977771887875/47872z23-6387511968556\
  275/909568z24
↦ j[43]=xy6z9-287/187y3z13-3294/1309y2z14-1985/1309yz15+24/1309z16+138510/5\
  39y2z15+138510/539yz16+138510/539z17-1420/1309yz20+112120/24871z21-190092\
  825/73304yz21+1846696725/146608z22+832426875/11968yz22+8758799679375/1172\
  864z23-8688391694008147 5/44568832z24
↦ j[44]=xz14+yz14+z15+5z20-151575/8z23+647139375/64z24
↦ j[45]=xyz13+y2z13+yz14+5yz19+23925/8yz22+43875/4z23-647139375/128z24
↦ j[46]=xy2z12+y3z12+y2z13-5yz19-5z20-23925/8yz22-8475z23+319696875/128z24
↦ j[47]=xy3z11+y4z11+y3z12+5z20+16875/8yz22+13575/2z23-155975625/128z24
↦ j[48]=y5z10+5/2y4z11+55/17y3z12+40/17y2z13+29/34yz14+2/17z15-438615/12716\
  y3z13-1315845/25432y2z14-1315845/25432yz15-438615/25432z16+905/102yz19+74\
  5/102z20+1707075/6358yz20+13407525/50864z21+2959284375/50864yz21+29592843\
  75/101728z22-7018033348175/406912yz22-7019757749975/813824z23+82848936375\
  /813824z24
```

```
↦ j[49]=xy4z10-3/2y4z11-55/17y3z12-40/17y2z13-29/34yz14-2/17z15+438615/1271\
  6y3z13+1315845/25432y2z14+1315845/25432yz15+438615/25432z16-395/102yz19-7\
  45/102z20-1707075/6358yz20-13407525/50864z21-2959284375/50864yz21-2959284\
  375/101728z22+7017533609375/406912yz22+7015637765975/813824z23+3391275498\
  75/813824z24
↦ j[50]=xy5z9+y6z9-5/2y4z11-55/17y3z12-40/17y2z13-29/34yz14-2/17z15+438615/\
  12716y3z13+1315845/25432y2z14+1315845/25432yz15+438615/25432z16-1415/102y\
  z19-1255/102z20-1707075/6358yz20-13407525/50864z21-2959284375/50864yz21-2\
  959284375/101728z22+7018189754975/406912yz22+7022473887575/813824z23-1707\
  20470125/813824z24
↦ j[51]=y7z8+7/2y6z9-35/4y4z11-2848/187y3z12-4617/374y2z13-3437/748yz14-115\
  /187z15+58551255/279752y3z13+175653765/559504y2z14+175653765/559504yz15+5\
  8551255/559504z16-38545/748yz19-34055/748z20-107445825/69938yz20-17378354\
  25/1119008z21-443675323125/1119008yz21-443675323125/2238016z22+5625720982\
  51725/8952064yz22+562790878490325/17904128z23-4547863128375/17904128z24
↦ j[52]=xy6z8-5/2y6z9+35/4y4z11+2848/187y3z12+4617/374y2z13+3437/748yz14+11\
  5/187z15-58551255/279752y3z13-175653765/559504y2z14-175653765/559504yz15-\
  58551255/559504z16+38545/748yz19+37795/748z20+107445825/69938yz20+1737835\
  425/1119008z21+443675323125/1119008yz21+443675323125/2238016z22-562568321\
  599725/8952064yz22-562820420301525/17904128z23+2614689385875/17904128z24
↦ j[53]=y8z7-878/141y6z9+9877/517y4z11+915172/26367y3z12+253943/8789y2z13+9\
  4823/8789yz14+37210/26367z15-847642725/1643543y3z13-2534922297/3287086y2z\
  14-2534922297/3287086yz15-839636847/3287086z16+5435755695/3503192y2z15+54\
  35755695/3503192yz16+5435755695/3503192z17+1063395/8789yz19+5880395/52734\
  z20+6007578570/1643543yz20+25156075905/6574172z21+1064589124796775/101242\
  2488yz21+1075884396851925/2024844976z22-2383632331137825/26296688yz22+103\
  195950091159325/1012422488z23-274018430271493924425/16198759808z24
↦ j[54]=xy7z7+769/282y6z9-21413/2068y4z11-30212/1551y3z12-17111/1034y2z13-1\
  2809/2068yz14-1235/1551z15+237013695/773432y3z13+707273613/1546864y2z14+7\
  07273613/1546864yz15+233246223/1546864z16-5435755695/3503192y2z15-5435755\
  695/3503192yz16-5435755695/3503192z17-133305/2068yz19-409355/6204z20-4097\
  17845/193358yz20-639413595/281248z21-156040912046925/238217056yz21-158698\
  623118725/476434112z22+688041879438525/24749824yz22-508310646770040175/38\
  11472896z23+644765667674533517 25/3811472896z24
↦ j[55]=xz13+yz13+z14-20007/154y2z15-20007/154yz16-20007/154z17+5z19-100035\
  /154yz21-1933395/308z22-4406592375/1232z23+576566423518975/561792z24
↦ j[56]=xyz12+y2z12+yz13+20007/308y2z15+20007/308yz16+20007/308z17+5yz18+20\
  42295/616yz21+5709015/1232z22+2581875/64yz22+17825173875/9856z23-57656547\
  5494975/1123584z24
↦ j[57]=xy2z11+y3z11+y2z12-13851/308y2z15-13851/308yz16-13851/308z17-5yz18-\
  5z19-1980735/616yz21-5038095/1232z22-2581875/64yz22-22282522875/9856z23+9\
  15874327042225/1123584z24
↦ j[58]=xy3z10+y4z10+y3z11+1539/44y2z15+1539/44yz16+1539/44z17+5z19+201015/\
  88yz21+594255/176z22+7745625/64yz22+3558400875/1408z23-77537732695475/802\
  56z24
↦ j[59]=xy4z9+y5z9+y4z10-7695/308y2z15-7695/308yz16-7695/308z17+5yz18-83347\
  5/616yz21-3235275/1232z22-12909375/64yz22-21828234375/9856z23+92636468662\
  2125/1123584z24
↦ j[60]=xy5z8+y6z8+y5z9+4617/308y2z15+4617/308yz16+4617/308z17-5yz18-5z19+2\
  82945/616yz21+2310765/1232z22+13820625/64yz22+15566792625/9856z23-6027924\
  06737525/1123584z24
↦ j[61]=xy6z7+y7z7+y6z8-1539/308y2z15-1539/308yz16-1539/308z17+5z19+244485/\
  616yz21-1432455/1232z22-10479375/64yz22-7060030875/9856z23+10437976377287\
  5/561792z24
↦ j[62]=y8z6+4y7z7+1096/141y6z8+438/47y5z9+3827/517y4z10+130/33y3z11+745/51\
  7y2z12+185/517yz13+74/1551z14-626565/48598y6z9+36597195/1069156y4z11+2938\
  45620/4543913y3z12+531935685/9087826y2z13+441719055/18175652yz14+30706050\
  /4543913z15-824462422425/617972168y3z13+11079250419375/13595387696y2z14+1\
  1079250419375/13595387696yz15+29217423712725/13595387696z16+6498875861433\
  477/3622300528y2z15+6498875861433477/3622300528yz16+6498875861433477/3622\
  300528z17-115/47yz18-715/282z19+568770225/1652332yz19+75725325/150212z20+\
  83154721712625/3398846924yz20+2507243939897625/27190775392z21+23491579901\
  268980805/2093689705184yz21+16367386229217586595/523422426296z22-13196224\
  488747115125/19775109376yz22+12686378658993706882l0625/8374758820736z23-1\
  7691473521227817276849545175/904473952639488z24
↦ j[63]=xy7z6-3y7z7-1096/141y6z8-438/47y5z9-3827/517y4z10-130/33y3z11-745/5\
  17y2z12-185/517yz13-74/1551z14+626565/48598y6z9-36597195/1069156y4z11-293\
  845620/4543913y3z12-531935685/9087826y2z13-441719055/18175652yz14-3070605\
  0/4543913z15+824462422425/617972168y3z13-11079250419375/13595387696y2z14-\
  11079250419375/13595387696yz15-29217423712725/13595387696z16-649889396117\
  5401/3622300528y2z15-6498893961175401/3622300528yz16-6498893961175401/362\
  2300528z17+350/47yz18+715/282z19-568770225/1652332yz19-75725325/150212z20\
  -83154721712625/3398846924yz20-2507243939897625/27190775392z21-2349420352\
  2192320265/2093689705184yz21-32734262338246025995/1046844852592z22+131967\
  40688623697625/19775109376yz22-634320342549593324852625/4187379410368z23+\
  336141189847485260140967097625/17185005100150272z24
↦ j[64]=y9z5-52/5y7z7-6904/235y6z8-10137/235y5z9-20582/517y4z10-1327/55y3z1\
  1-25293/2585y2z12-7006/2585yz13-1072/2585z14+5639085/97196y6z9-329374755/\
  2138312y4z11-1284909129/4543913y3z12-4563044199/18175652y2z13-3526717563/\
  36351304yz14-238958289/9087826z15+7263199915155/1235944336y3z13-104892996\
  034485/27190775392y2z14-104892996034485/27190775392yz15-264683394167895/2\
  7190775392z16-292449214667345301/36223005280y2z15-292449214667345301/3622\
  3005280yz16-292449214667345301/36223005280z17+377/94yz18+689/47z19-491972\
  4405/3304664yz19-7499192895/3304664z20-465864054810075/3398846924yz20-232\
  72639204604175/54381550784z21-210513611918733814743/4187379410368yz21-588\
  763946311409477793/4187379410368z22+10568763114146744625/3595474432yz22-2\
  2836546608337302729045625/33499035282944z23+33613907149852300781381322425\
  5/3818890022255616z24
↦ j[65]=xy8z5+32/5y7z7+15232/705y6z8+7947/235y5z9+16755/517y4z10+3331/165y3\
  z11+21568/2585y2z12+6081/2585yz13+2846/7755z14-4385955/97196y6z9+25618036\
  5/2138312y4z11+991063509/4543913y3z12+3499172829/18175652y2z13+2643279453\
  /36351304yz14+177546189/9087826z15-510388642755/112358576y3z13+8273449519\
  5735/27190775392y2z14+82734495195735/27190775392yz15+206248546742445/2719\
  0775392z16+227460999045268251/36223005280y2z15+227460999045268251/3622300\
  5280yz16+227460999045268251/36223005280z17-617/94yz18-4829/282z19+3438349\
  05/300424yz19+5833235745/3304664z20+191354666548725/1699423462yz20+182581\
  51324808925/54381550784z21+163539441692436671493/4187379410368yz21+457825\
  644211426125893/4187379410368z22-89855404516525595625/39550218752yz22+177\
  62048053885050003338625/33499035282944z23-2352994088872107067347530909545\
  /34370010200300544z24
↦ j[66]=y10z4+22y7z7+10142/141y6z8+5489/47y5z9+5475/47y4z10+226/3y3z11+1503\
  /47y2z12+436/47yz13+212/141z14-970935/8836y6z9+54553185/194392y4z11+20150\
  2425/413083y3z12+709272945/1652332y2z13+491141745/3304664yz14+42414045/82\
  6166z15-13268790182475/1235944336y3z13+326186439486885/17303220704y2z14+3\
  26186439486885/17303220704yz15+511949502041535/17303220704z16+20533149915\
  4193607/9220401344y2z15+205331499154193607/9220401344yz16+205331499154193\
    607/9220401344z17+1015/94yz18-11195/282z19+9119149125/3304664yz19+1612626\
    9975/3304664z20+1722252380149575/4325805176yz20+40486832246297475/3460644\
    1408z21+86222628281316862035/666173997104yz21+3827600153784459109735/1065\
    8783953664z22-212431074992817667875/39550218752yz22+162431268984322756306\
    439625/85270271629312z23-5310151622816439181656857454575/2187182467291852\
    8z24
↦ j[67]=y11z3-44y7z7-21428/141y6z8-12122/47y5z9-12530/47y4z10-532/3y3z11-35\
    81/47y2z12-1058/47yz13-536/141z14+640905/8836y6z9-25563555/194392y4z11-35\
    2065/2209y3z12-1435545/8836y2z13-6018435/194392yz14-334095/4418z15+453354\
    657075/72702608y3z13-67922428047615/1017836512y2z14-67922428047615/101783\
    6512yz15-74269393246665/1017836512z16-893884195353597003/18440802688y2z15\
    -893884195353597003/18440802688yz16-893884195353597003/18440802688z17-283\
    5/47yz18+22385/282z19-395171175/194392yz19-1172470725/194392z20-360596839\
    248225/508918256yz20-4892289394562775/2035673024z21-79258578474346178505/\
    313493645696yz21-86160351686711096894 5/1253974582784z22+7271656811928160 8\
    75/2326483456yz22-424113915102229131954243 75/10031796662272z23+6799137804\
    1707297319736831582 5/1286577921936384z24
↦ j[68]=xy10z3+22y7z7+3762/47y6z8+6633/47y5z9+7055/47y4z10+102y3z11+2078/47\
    y2z12+622/47yz13+108/47z14+165015/4418y6z9-14494815/97196y4z11-135666270/\
    413083y3z12-220413015/826166y2z13-194414175/1652332yz14+10030860/413083z1\
    5+695220126525/154493042y3z13+3765885624193 5/786510032y2z14+376588562419 3\
    5/786510032yz15+375315091575885/8651610352z16+483221842054194717/18440802\
    688y2z15+483221842054194717/18440802688yz16+483221842054194717/184408026 8\
    8z17+5125/94yz18-1865/47z19-1200619575/1652332yz19+1902866175/1652332z20+\
    268564150692 0675/8651610352yz20+10670521865317425/8651610352z21+657636174\
    738694524465/5329391976832yz21+699209468404526430043 5/21317567907328z22+5\
    553968548621501125/2471888672yz22+396131943695600228635027125/17054054325\
    8624z23-1041400629215382738611919651175/3645304112153088z24
↦ j[69]=xz12+yz12+z13+572/423y6z9-640/141y4z11-62420/7191y3z12-18392/2397y2\
    z13-2504/799yz14-2480/7191z15+1900665/13583y3z13+361663461/2091782y2z14+3\
    61663461/2091782yz15+68961051/2091782z16-8853766965/1114652y2z15-88537669\
    65/1114652yz16-8853766965/1114652z17+5z18-212420/7191yz19-68020/2397z20-1\
    332663435/1045891yz20-8824397565/4183564z21-44412306927525/161067214yz21-\
    196723166744925/322134428z22+91232350818875/1303968yz22-85182191428382527\
    225/293786598336z23+270414402753817556842 5/20616603392z24
↦ j[70]=xyz11+y2z11+yz12-286/423y6z9+320/141y4z11+31210/7191y3z12+9196/2397\
    y2z13+1252/799yz14+1240/7191z15-15577245/149413y3z13-144213534/1045891y2z\
    14-144213534/1045891yz15-3197529/95081z16+8853766965/2229304y2z15+8853766\
    965/2229304yz16+8853766965/2229304z17+5yz17+106210/7191yz19+34010/2397z20\
    +240968535/190162yz20+1567041480/1045891z21+327495838518225/1288537712yz2\
    1+936739277787825/2577075424z22-41631712161551725/1090117248yz22+16848071\
    263565223392 5/1175146393344z23-270416438435177923342 5/41233206784z24
↦ j[71]=xy2z10+y3z10+y2z11+286/423y6z9-320/141y4z11-31210/7191y3z12-9196/23\
    97y2z13-1252/799yz14-1240/7191z15+15577245/149413y3z13+23588253/149413y2z\
    14+23588253/149413yz15+8011008/149413z16-2186449605/318472y2z15-218644960\
    5/318472yz16-2186449605/318472z17-5yz17-5z18-106210/7191yz19-34010/2397z2\
    0-348801645/298826yz20-284992095/298826z21-49448618214075/184076816yz21-1\
    96416565735275/368153632z22+41631071717668525/1090117248yz22-245593810261\
    51879675/167878056192z23+30964629551859158422 5/5890458112z24
↦ j[72]=xy3z9+y4z9+y3z10-286/423y6z9+320/141y4z11+31210/7191y3z12+9196/2397\
    y2z13+1252/799yz14+1240/7191z15-13527810/149413y3z13-154050822/1045891y2z\
    14-154050822/1045891yz15-59356152/1045891z16+9265418685/1114652y2z15+9265\
    418685/1114652yz16+9265418685/1114652z17+5z18+106210/7191yz19+34010/2397z\
    20+1234402740/1045891yz20+1383637755/2091782z21+105070137902325/322134428\
    yz21+414705570236775/644268856z22-6168510692255975/136264656yz22+66200788\
    7526707950/4590415599z23-949601768600740724325/20616603392z24
 ↦ j[73]=xy4z8+y5z8+y4z9+286/423y6z9-320/141y4z11-31210/7191y3z12-9196/2397y\
    2z13-1252/799yz14-1240/7191z15+11478375/149413y3z13+142983873/1045891y2z1\
    4+142983873/1045891yz15+62635248/1045891z16-13234976775/2229304y2z15-1323\
    4976775/2229304yz16-13234976775/2229304z17+5yz17-106210/7191yz19-34010/23\
    97z20-2495999445/2091782yz20-31540410/95081z21-469793494110375/1288537712\
    yz21-1411221542610375/2577075424z22+57065153864289475/1090117248yz22-1254\
    8681666592952425/391715464448z23+696060504549723377475/41233206784z24
 ↦ j[74]=xy5z7+y6z7+y5z8-286/423y6z9+320/141y4z11+31210/7191y3z12+9196/2397y\
    2z13+1252/799yz14+1240/7191z15-9428940/149413y3z13-131916924/1045891y2z14\
    -131916924/1045891yz15-65914344/1045891z16+3678340815/2229304y2z15+367834\
    0815/2229304yz16+3678340815/2229304z17-5yz17-5z18+106210/7191yz19+34010/2\
    397z20+2601635235/2091782yz20+4140285/2091782z21+421375504864725/12885377\
    12yz21+812249983763325/2577075424z22-79973440545621125/1090117248yz22-167\
    063188394749078475/1175146393344z23+974495213822460648825/41233206784z24
 ↦ j[75]=xy6z6+y7z6+y6z7+286/423y6z9-320/141y4z11-31210/7191y3z12-9196/2397y\
    2z13-1252/799yz14-1240/7191z15+7379505/149413y3z13+120849975/1045891y2z14\
    +120849975/1045891yz15+69193440/1045891z16+4568267565/1114652y2z15+456826\
    7565/1114652yz16+4568267565/1114652z17+5z18-106210/7191yz19-34010/2397z20\
    -1392856425/1045891yz20+607166625/2091782z21-138961955560275/644268856yz2\
    1+3246371081325/1288537712z22+59036431911624575/545058624yz22+42739607320\
    93934325/10308301696z23-424081801037561315625/5154150848z24
 ↦ j[76]=xy7z5+y8z5+y7z6-286/423y6z9+320/141y4z11+31210/7191y3z12+9196/2397y\
    2z13+1252/799yz14+1240/7191z15-5330070/149413y3z13-109783026/1045891y2z14\
    -109783026/1045891yz15-72472536/1045891z16-1123108335/101332y2z15-1123108\
    335/101332yz16-1123108335/101332z17+5yz17+106210/7191yz19+34010/2397z20+2\
    969790465/2091782yz20-570015855/1045891z21+373302122175/29284948yz21-2281\
    0656626775/58569896z22-50436732640756175/272529312yz22-545752583844195702\
    5/6676968144z23+608955833646803601675/3748473344z24
 ↦ j[77]=y9z4+9/2y8z5+38/5y7z6+28/5y6z7-6/5y5z8-13/2y4z9-32/5y3z10-33/10y2z1\
    1-11/10yz12-1/5z13-243/10y7z7-1701/20y6z8-1863/20y5z9-81/4y4z10+1539/20y3\
    z11+1863/20y2z12+891/20yz13+81/10z14-1573/2115y6z9+352/141y4z11-157814630\
    /79101y3z12-789848639/263670y2z13-263502293/87890yz14-158162263/158202z15\
    +403948702323/13148344y3z13+8474527251387/184076816y2z14+8474527251387/18\
    4076816yz15+2819245418865/184076816z16-2277893907/253330y2z15-2277893907/\
    253330yz16-2277893907/253330z17-7yz17+13/4z18+486yz18+1701/4z19-231480772\
    3/158202yz19+5037026/26367z20+603143500229535/92038408yz20+11634440010787\
    95/368153632z21-134266161760124013/2577075424yz21-137028452905078101/5154\
    150848z22+1114271066009961257735/71947738368yz22+51874457100139726934015/\
    7050878360064z23+276524393448922094247795/1566861857792z24
 ↦ j[78]=xy8z4-7/2y8z5-38/5y7z6-28/5y6z7+6/5y5z8+13/2y4z9+32/5y3z10+33/10y2z\
    11+11/10yz12+1/5z13+243/10y7z7+1701/20y6z8+1863/20y5z9+81/4y4z10-1539/20y\
    3z11-1863/20y2z12-891/20yz13-81/10z14+1001/705y6z9-224/47y4z11+52490440/2\
    6367y3z12+262945693/87890y2z13+263364573/87890yz14+52711661/52734z15-4036\
    60006443/13148344y3z13-8457153221835/184076816y2z14-8457153221835/1840768\
    16yz15-2805913131633/184076816z16+317919689883/11146520y2z15+317919689883\
    /11146520yz16+317919689883/11146520z17+2yz17-33/4z18-486yz18-1701/4z19+77\
    0823701/52734yz19-1803712/8789z20-603278818984755/92038408yz20-1163149571\
    242875/368153632z21+135044223950364363/2577075424yz21+141273988656213051/\
    5154150848z22-1090420035071554691885/71947738368yz22-41792722244763474154\
    565/7050878360064z23-685099534954253407593345/1566861857792z24
```

```
↦ j[79]=xy9z3+y10z3-9/2y8z5-38/5y7z6-28/5y6z7+6/5y5z8+13/2y4z9+32/5y3z10+33\
  /10y2z11+11/10yz12+1/5z13+243/10y7z7+1701/20y6z8+1863/20y5z9+81/4y4z10-15\
  39/20y3z11-1863/20y2z12-891/20yz13-81/10z14+143/2115y6z9-32/141y4z11+1581\
  57940/79101y3z12+790860199/263670y2z13+263640013/87890yz14+158189543/1582\
  02z15-404057047923/13148344y3z13-8489953497915/184076816y2z14-84899534979\
  15/184076816yz15-2833154826993/184076816z16-232919344017/11146520y2z15-23\
  2919344017/11146520yz16-232919344017/11146520z17+7yz17+7/4z18-486yz18-170\
  1/4z19+2317144343/158202yz19-4662916/26367z20-603006984939855/92038408yz2\
  0-1163846020930875/368153632z21+132236704098135963/2577075424yz21+1303312\
  19207468751/5154150848z22-1159686946509974573885/71947738368yz22-68677251\
  539404951638665/7050878360064z23+303844033154752813237905/1566861857792z2\
  4
↦ j[80]=y11z2+11/2y10z3-165/4y8z5-88y7z6-77y6z7+275/4y4z9+149/2y3z10+161/4y\
  2z11+55/4yz12+5/2z13+1701/4y7z7+11907/8y6z8+13041/8y5z9+2835/8y4z10-10773\
  /8y3z11-13041/8y2z12-6237/8yz13-567/4z14+7865/846y6z9-4400/141y4z11+27636\
  44230/79101y3z12+5531165905/105468y2z13+1844819035/35156yz14+5535979285/3\
  16404z15-14155627908105/26296688y3z13-297065656926729/368153632y2z14-2970\
  65656926729/368153632yz15-98886866213259/368153632z16+467371398141/445860\
  8y2z15+467371398141/4458608yz16+467371398141/4458608z17+100yz17-275/8z18-\
  8505yz18-59535/8z19+81043973125/316404yz19-86090350/26367z20-211109592183\
  47385/184076816yz20-40725259662179865/736307264z21+4706734673877922755/51\
  54150848yz21+4766400819649905885/10308301696z22-2049414034886720541775/75\
  73446144yz22-1836647027390449275817975/14101756720128z23-6941017444566613\
  228827975/3133723715584z24
↦ j[81]=xy10z2-9/2y10z3+165/4y8z5+88y7z6+77y6z7-275/4y4z9-149/2y3z10-161/4y\
  2z11-55/4yz12-5/2z13-1701/4y7z7-11907/8y6z8-13041/8y5z9-2835/8y4z10+10773\
  /8y3z11+13041/8y2z12+6237/8yz13+567/4z14-2431/282y6z9+1360/47y4z11-921329\
  180/26367y3z12-1843856843/35156y2z13-1844874123/35156yz14-1845344615/1054\
  68z15+14155483898745/26296688y3z13+297092613853737/368153632y2z14+2970926\
  13853737/368153632yz15+98915839271307/368153632z16-270887899131/4458608y2\
  z15-270887899131/4458608yz16-270887899131/4458608z17-95yz17+275/8z18+8505\
  yz18+59535/8z19-27016215455/105468yz19+28572080/8789z20+21110683794699105\
  /184076816yz20+40726306493438745/736307264z21-4698753505995194955/5154150\
  848yz21-4748618833102825785/10308301696z22+39114939183464627699725/143895\
  476736yz22+1893501137798116652057275/14101756720128z23+536564751106962074\
  4877275/3133723715584z24
↦ j[82]=xz11+yz11+z12+572/423y6z8+572/141y5z9+790/141y4z10+40/9y3z11+88/47y\
  2z12+46/141yz13+56/423z14-582770/6627y6z9+5608430/24299y4z11+540007510/12\
  39249y3z12+140696995/413083y2z13+45353685/413083yz14-8938535/1239249z15-1\
  27805831775/14044822y3z13-26678525255325/2162902588y2z14-26678525255325/2\
  162902588yz15-6996427161975/2162902588z16+965701597805963/1152550168y2z15\
  +965701597805963/1152550168yz16+965707360556803/1152550168z17+910/141yz18\
  +530/423z19+235982375/112659yz19+147407075/75106z20+41856813119250/540725\
  647yz20+466506627125625/4325805176z21+813428824867634820/41635874819yz21+\
  360803350499543037395/18985958917464z22-2044981435549759625/449434304yz22\
  +379820494972020072979875/5329391976832z23-74930243087916747964230028075/\
  8201934252344448z24
↦ j[83]=xyz10+y2z10+yz11-44/3y7z7-22000/423y6z8-12694/141y5z9-4440/47y4z10-\
  572/9y3z11-1266/47y2z12-368/47yz13-592/423z14+291385/6627y6z9-2804215/242\
  99y4z11-354995030/1239249y3z12-112844135/413083y2z13-65172480/413083yz14-\
  38026370/1239249z15-4104216934875/154493042y3z13-87610866574725/216290258\
  8y2z14-87610866574725/2162902588yz15-30151829486475/2162902588z16-9657361\
  51858727/2305100336y2z15-965724626357047/2305100336yz16-965736151858727/2\
   305100336z17-3980/141yz18+10310/423z19-3747910075/2478498yz19-395139025/4\
   13083z20-1480535146859625/1081451294yz20-2605859335031625/4325805176z21+4\
   9514451750738614374865/75943835669856yz21+16271513774910759142135/5062922\
   3779904z22+11326226929216215783875/19775109376yz22+532079018479511834489\
   9625/21317567907328z23+7498612077114066743859047492 5/16403868504688896z24
↦ j[84]=xy2z9+y3z9+y2z10+44/3y7z7+22000/423y6z8+12694/141y5z9+4440/47y4z10+\
   572/9y3z11+1266/47y2z12+368/47yz13+592/423z14+432415/6627y6z9-5668240/242\
   99y4z11-469386380/1239249y3z12-120929870/413083y2z13-24569790/413083yz14+\
   25735165/1239249z15+5850299388375/154493042y3z13+125029896376005/21629025\
   88y2z14+125029896376005/2162902588yz15+43125704938755/2162902588z16+14758\
   30228106393/2305100336y2z15+1475818702604713/2305100336yz16+1475818702604\
   713/2305100336z17+3980/141yz18-10310/423z19-2165535925/2478498yz19-919789\
   925/826166z20+1387273664409075/1081451294yz20+2276001050636925/4325805176\
   z21-50878211929841413645085/75943835669856yz21-16566371038690273007315/50\
   629223779904z22-11214474564379130063875/19775109376yz22-4861138346456520 2\
   15783625/21317567907328z23-11456670986606800634488973942 5/164038685046888\
   96z24
↦ j[85]=xy3z8+y4z8+y3z9-44/3y7z7-22000/423y6z8-12694/141y5z9-4440/47y4z10-5\
   72/9y3z11-1266/47y2z12-368/47yz13-592/423z14-794315/6627y6z9+19808935/485\
   98y4z11+898575340/1239249y3z12+246316000/413083y2z13+155880105/826166yz14\
   -11090435/1239249z15-13583602174275/308986084y3z13-290354162377815/432580\
   5176y2z14-290354162377815/4325805176yz15-100183731937965/4325805176z16-86\
   5434793775917/1152550168y2z15-865434793775917/1152550168yz16-865429031025\
   077/1152550168z17-3980/141yz18+10310/423z19+5278324775/2478498yz19+879169\
   625/413083z20-659615379073650/540725647yz20-4122118457610525/8651610352z2\
   1+25548497753349154062865/37971917834928yz21+82797198028512363 13835/25314\
   611889952z22+55120018176135629683 75/9887554688yz22+22793867670169023814 10\
   625/10658783953664z23+13435394852371601137045685792 5/16403868504688896z24
↦ j[86]=xy4z7+y5z7+y4z8+44/3y7z7+22000/423y6z8+12694/141y5z9+4440/47y4z10+5\
   72/9y3z11+1266/47y2z12+368/47yz13+592/423z14+820165/6627y6z9-10004695/242\
   99y4z11-924375050/1239249y3z12-252843830/413083y2z13-82764015/413083yz14+\
   12472705/1239249z15+6922621646775/154493042y3z13+149031192290445/21629025\
   88y2z14+149031192290445/2162902588yz15+52114489235595/2162902588z16+24823\
   81227491245/2305100336y2z15+2482392752992925/2305100336yz16+2482381227491\
   245/2305100336z17+3980/141yz18-10310/423z19-5645585125/2478498yz19-843374\
   425/413083z20+1297187944049925/1081451294yz20+2062173912456825/4325805176\
   z21-50561824847932108997975/75943835669856yz21-16202257051507079051225/50\
   629223779904z22-10885418295936912302125/19775109376yz22-3872765189989082 1\
   41649375/21317567907328z23-19265698366370529429928517042 5/164038685046888\
   96z24
↦ j[87]=xy5z6+y6z6+y5z7-44/3y7z7-22000/423y6z8-12694/141y5z9-4440/47y4z10-5\
   72/9y3z11-1266/47y2z12-368/47yz13-592/423z14-677990/6627y6z9+16073845/485\
   98y4z11+748480135/1239249y3z12+199942510/413083y2z13+126629655/826166yz14\
   -21868475/1239249z15-12867082326225/308986084y3z13-280465993175625/432580\
   5176y2z14-280465993175625/4325805176yz15-100326840608475/4325805176z16-34\
   82129007677641/2305100336y2z15-3482140533179321/2305100336yz16-3482140533\
   179321/2305100336z17-3980/141yz18+10310/423z19+2366512450/1239249yz19+587\
   789600/413083z20-1266613569572625/1081451294yz20-4261564971336375/8651610\
   352z21+16119256475079729722865/25314611889952yz21+4571793346822081749251 5\
   /151887671339712z22+10507480887510632554625/19775109376yz22+2851941890352\
   553038384375/21317567907328z23+27021836432967149090589823742 5/16403868504\
   688896z24
↦ j[88]=xy6z5+y7z5+y6z6+44/3y7z7+22000/423y6z8+12694/141y5z9+4440/47y4z10+5\
   72/9y3z11+1266/47y2z12+368/47yz13+592/423z14+502210/6627y6z9-5655550/2429\
   9y4z11-532246295/1239249y3z12-135155360/413083y2z13-39011010/413083yz14+3\
   2866945/1239249z15+2824416045900/77246521y3z13+31908535379955/540725647y2\
   z14+31908535379955/540725647yz15+12137623058655/540725647z16+305826537280\
   3955/1152550168y2z15+3058265372803955/1152550168yz16+3058271135554795/115\
   2550168z17+3980/141yz18-10310/423z19-1819427750/1239249yz19-607474925/826\
   166z20+607471821367650/540725647yz20+579107988351675/1081451294z21-222626\
   58475598588445775/37971917834928yz21-821349300336923555475/3164326486244z\
   22-4934719312810096564625/9887554688yz22-13859743719993905654625/66617399\
   7104z23-474598588592475460844829069175/16403868504688896z24
↦ j[89]=xy7z4+y8z4+y7z5-44/3y7z7-22000/423y6z8-12694/141y5z9-4440/47y4z10-5\
   72/9y3z11-1266/47y2z12-368/47yz13-592/423z14-360035/6627y6z9+670505/4418y\
   4z11+32395580/112659y3z12+7477640/37553y2z13+3556695/75106yz14-3842065/11\
   2659z15-834107335425/28089644y3z13-20483018940645/393255016y2z14-20483018\
   940645/393255016yz15-8805516244695/393255016z16-62947700852446/13097161y2\
   z15-62947635366641/13097161yz16-62947700852446/13097161z17-3980/141yz18+1\
   0310/423z19+128147225/112659yz19+14387975/75106z20-98863714608525/9831375\
   4yz20-465840806838075/786510032z21+1115523758314184084115/2301328353632yz\
   21+605939354848035542885/3451992530448z22+766419965297948840125/179773721\
   6yz22-189306790888410903168375/968980359424z23+39071746921686379894793389\
   525/745630386576768z24
↦ j[90]=xy8z3+y9z3+y8z4+44/3y7z7+22000/423y6z8+12694/141y5z9+4440/47y4z10+5\
   72/9y3z11+1266/47y2z12+368/47yz13+592/423z14+385885/6627y6z9-3788005/2429\
   9y4z11-382151090/1239249y3z12-88781870/413083y2z13-24385785/413083yz14+43\
   644985/1239249z15+1715524031025/77246521y3z13+102607196576895/2162902588y\
   2z14+102607196576895/2162902588yz15+54572523708195/2162902588z16+41491904\
   700872569/4610200672y2z15+41491881649869209/4610200672yz16+41491881649869\
   209/4610200672z17+3980/141yz18-10310/423z19-3279442975/2478498yz19-148639\
   775/826166z20+1653946905334725/2162902588yz20+1460308283586225/2162902588\
   z21-11411768595004261871945/37971917834928yz21-10977086845197539633785/30\
   3775342679424z22-5639977208141893583875/19775109376yz22+26988004909046083\
   142851125/42635135814656z23-1609576594686224917280791595425/1640386850468\
   8896z24
↦ j[91]=xy9z2+y10z2+y9z3-44/3y7z7-22000/423y6z8-12694/141y5z9-4440/47y4z10-\
   572/9y3z11-1266/47y2z12-368/47yz13-592/423z14-747785/6627y6z9+16048465/48\
   598y4z11+811340050/1239249y3z12+214168000/413083y2z13+155512095/826166yz1\
   4-29000255/1239249z15-1869344233875/154493042y3z13-27348503342535/6179721\
   68y2z14-27348503342535/617972168yz15-19871126407035/617972168z16-21906823\
   763821361/1317200192y2z15-21906823763821361/1317200192yz16-21906817177820\
   401/1317200192z17-3980/141yz18+10310/423z19+6392231825/2478498yz19+493594\
   550/413083z20-80051332955175/617972168yz20-220738142474775/308986084z21-2\
   3878555064365101845/380670855488yz21-17328862007227564772735/867929550512\
   64z22-1868878877799862375/308986084yz22-17995252524739734383894625/121814\
   67375616z23+424904594031505848545727956275/2343409786384128z24
↦ j[92]=y11z+11/2y10z2+35/2y9z3+75/2y8z4+45y7z5+21y6z6-21y5z7-45y4z8-75/2y3\
   z9-35/2y2z10-11/2yz11-z12+242/3y7z7+117854/423y6z8+66671/141y5z9+68915/14\
   1y4z10+2926/9y3z11+6479/47y2z12+5819/141yz13+2948/423z14-14820025/17672y6\
   z9+873381255/388784y4z11+3434435885/826166y3z12+10920930675/3304664y2z13+\
   6994380015/6609328yz14-22734095/1652332z15+146778297314625/2471888672y3z1\
   3+3946648865169015/34606441408y2z14+3946648865169015/34606441408yz15+1719\
   77518433115/3146040128z16+586848438081958029/36881605376y2z15+58684760824\
   5837069/36881605376yz16+586849267918078989/36881605376z17+16885/141yz18-5\
   9620/423z19+128385353575/6609328yz19+137287540725/6609328z20+127263206240\
   91225/1573020064yz20+310587762800398275/69212882816z21-559423970391860132\
   618355/202516895119616yz21-2648529752258055957642185/2430202741435392z22-\
   2062519266963263625311125/79100437504yz22-6368843586601514282682375/341081\
   086517248z23-22766825225665150008965210140975/131230948037511168z24
↦  j[93]=xy10z-9/2y10z2-35/2y9z3-75/2y8z4-45y7z5-21y6z6+21y5z7+45y4z8+75/2y3\
   z9+35/2y2z10+11/2yz11+z12-66y7z7-95854/423y6z8-53977/141y5z9-55595/141y4z\
   10-2354/9y3z11-5213/47y2z12-4715/141yz13-2356/423z14+46829465/53016y6z9-9\
   39859165/388784y4z11-11220311425/2478498y3z12-11824529065/3304664y2z13-76\
   71452325/6609328yz14+325274275/4956996z15-213307344103275/2471888672y3z13\
   -3539899421796285/34606441408y2z14-3539899421796285/34606441408yz15-55359\
   6604350435/34606441408z16+637916404054624945/36881605376y2z15+63791741829\
   8772785/36881605376yz16+637915574218503985/36881605376z17-12905/141yz18+4\
   9310/423z19-399356020775/19827984yz19-125295168175/6609328z20-16067719441\
   9307025/17303220704yz20-248929945229340225/69212882816z21+217687447956085\
   6008821875/607550685358848yz21+1506720720000815632794625/810067580478464z\
   22+251542729646597163392375/79100437504yz22+1100231254123070657862622125/\
   341081086517248z23-24743902157297674796181020605675/131230948037511168z24
↦  j[94]=xz10+yz10+z11-2y10z3+55/3y8z5+88/3y7z6-176/3y5z8-275/3y4z9-74y3z10-\
   107/3y2z11-35/3yz12-2z13-243y7z7-269321/282y6z8-117041/94y5z9-712895/1034\
   y4z10+21007/66y3z11+646581/1034y2z12+323287/1034yz13+26296/1551z14+153218\
   6301053/31491504y6z9-28302886184245/230937696y4z11-369236982688165/147222\
   7812y3z12-483241821082753/1962970416y2z13-161778192344447/1308646944yz14-\
   129648185396255/2944455624z15+26412860150841105/4943777344y3z13-695810883\
   7563240127/761341710976y2z14-6958108837563240127/761341710976yz15-1102568\
   5494084215417/761341710976z16-8866818239102802159903/811395318272y2z15-88\
   66818239102802159903/811395318272yz16-8866818239102802159903/811395318272\
   z17-60yz17+205/6z18+204970/47yz18+481415/141z19-1628796093113185/10707111\
   36yz19-801624018015785/356903712z20-22622732585079460105/380670855488yz20\
   -14651113564696435690405/28930985017088z21-14925866713467095080947885/234\
   493246980608yz21-162988809923300115947069165/937972987922432z22+217027488\
   303727311617857225/81157048879104yz22-3607577023721769331157959473779675/\
   3849441142433660928z23+114652925676746336308634050779694625/9623602856084\
   15232z24
↦  j[95]=xyz9+y2z9+yz10+y10z3-55/6y8z5-88/3y7z6-154/3y6z7-176/3y5z8-275/6y4z\
   9-73/3y3z10-49/6y2z11-11/6yz12-1/3z13+27807/2y7z7+27474989/564y6z8+157748\
   09/188y5z9+181916225/2068y4z10+7831277/132y3z11+52555821/2068y2z12+154100\
   57/2068yz13+3970631/3102z14-1532512780349/62983008y6z9+28310922597685/461\
   875392y4z11+312266920336765/2944455624y3z12+369221219233489/3925940832y2z\
   13+85718918102111/2617293888yz14+72587933139335/5888911248z15-23463796814\
   766465/9887554688y3z13+7638777282737732143/1522683421952y2z14+76387848961\
   54841903/1522683421952yz15+11252201992211767753/1522683421952z16+88666486\
   91778875305887/1622790636544y2z15+8866648691778875305887/1622790636544yz1\
   6+8866648691778875305887/1622790636544z17+5yz17+95/12z18+2454995/94yz18-1\
   1751445/564z19+1325039781765505/2141422272yz19+802915233344105/713807424z\
   20+212928577519537578635/2284025132928yz20+49269817023572113303135/173585\
   910102528z21+14679874250853206902545165/468986493961216yz21+1624709876091\
   88447706426285/1875945975844864z22-3390929639575960014605825/284761575014\
   4yz22+133634634272595832503329684033025/285143788328419328z23-11465538424\
   9151618443278198421108625/1924720571216830464z24
↦  j[96]=xy2z8+y3z8+y2z9-y10z3+55/6y8z5+88/3y7z6+154/3y6z7+176/3y5z8+275/6y4\
   z9+73/3y3z10+49/6y2z11+11/6yz12+1/3z13-27807/2y7z7-27413389/564y6z8-15713\
   209/188y5z9-180943285/2068y4z10-7779157/132y3z11-52131801/2068y2z12-15281\
   377/2068yz13-3881821/3102z14+1503720099149/62983008y6z9-27852903658885/46\
   1875392y4z11-306823078233565/2944455624y3z12-361692515570689/3925940832y2\
   z13-83295222980111/2617293888yz14-69490446159335/5888911248z15+2299615012\
   4279505/9887554688y3z13-7261203230864264191/1522683421952y2z14-7261210844\
   281373951/1522683421952yz15-10802617963420417721/1522683421952z16-8536928\
   288202947267487/1622790636544y2z15-8536928288202947267487/1622790636544yz\
   16-8536928288202947267487/1622790636544z17-5yz17-95/12z18-2405995/94yz18+\
   11933795/564z19-1293458993895505/2141422272yz19-781880589098105/713807424\
   z20-208445445580800540395/2284025132928yz20-4759487316259587125249 5/17358\
   5910102528z21-1416601645746450385187716 5/468986493961216yz21-156896085683\
   398800539506285/1875945975844864z22+302143100233275749845075/258874159104\
   yz22-128603398089795292365124010873025/2851437883284193 28z23+581010345224\
   7913696710282640471675/10130108269562265 6z24
↦ j[97]=x2yz8-y3z8-2y2z9-yz10-30800/141y6z8-30800/47y5z9-486470/517y4z10-26\
   060/33y3z11-212010/517y2z12-64340/517yz13-88810/1551z14+22216575/24299y6z\
   9-1060229025/534578y4z11-16801981800/4543913y3z12-17427554775/4543913y2z1\
   3-16831216125/9087826yz14-4780072500/4543913z15+29230743643155/308986084y\
   3z13-3371148939164373/6797693848y2z14-3371148939164373/6797693848yz15-401\
   4157322375303/6797693848z16-367990648300524555/905575132y2z15-36799064830\
   0524555/905575132yz16-367990648300524555/905575132z17-49000/47yz18-91175/\
   141z19-24367891875/826166yz19-4426482375/75106z20-6671312561017835/169942\
   3462yz20-2623685417737941 55/13595387696z21-22939678504419940497 75/1046844\
   852592yz21-12443823743852234750425/2093689705184z22+46770011788812019312 5\
   /9887554688yz22-5910757025839607943762693 75/16749517641472z23+23791422051\
   6043124513022246107 5/537031409379696z24
↦ j[98]=xy3z7+y4z7+y3z8+y10z3-55/6y8z5-88/3y7z6-154/3y6z7-176/3y5z8-275/6y4\
   z9-73/3y3z10-49/6y2z11-11/6yz12-1/3z13+27495/2y7z7+27074645/564y6z8+15501\
   365/188y5z9+16209175/188y4z10+696115/12y3z11+4659915/188y2z12+1367575/188\
   yz13+345895/282z14-135393068959/5725728y6z9+2511263108135/41988672y4z11+2\
   7678604351535/267677784y3z12+32605013191739/356903712y2z13+7506184515061/\
   237935808yz14+6209563032205/535355568z15-22781207413239945/9887554688y3z1\
   3+642550895521998149/138425765632y2z14+642550895521998149/138425765632yz1\
   5+961488491436185539/138425765632z16+7610974544222347601 25/147526421504y2\
   z15+761097454422234760125/147526421504yz16+761097454422234760125/14752642\
   1504z17+5yz17+95/12z18+1169950/47yz18-11997475/564z19+1279604898026905/21\
   41422272yz19+771355537242305/713807424z20+186618026970712655 05/2076386484\
   48yz20+424759536272867828 6405/15780537282048z21+1264608268419656004216375\
   /4263513581465 6yz21+1401021058181686753510637 5/170540543258624z22-3292545\
   274303768691510825/2847615750144yz22+114625136389334974428235847502 75/259\
   22162575310848z23-98418387502742046009846987384424 75/17497459738334822 4z2\
   4
↦ j[99]=xy4z6+y5z6+y4z7-y10z3+55/6y8z5+88/3y7z6+154/3y6z7+176/3y5z8+275/6y4\
   z9+73/3y3z10+49/6y2z11+11/6yz12+1/3z13-27183/2y7z7-26764501/564y6z8-15318\
   121/188y5z9-176093065/2068y4z10-7557373/132y3z11-50531529/2068y2z12-14830\
   573/2068yz13-3743269/3102z14+1405719171149/62983008y6z9-26098701044485/46\
   1875392y4z11-286531489029805/2944455624y3z12-336342178241569/3925940832y2\
   z13-76335479575631/2617293888yz14-62885031805175/5888911248z15+2144197673\
   2934385/9887554688y3z13-6602383295616654047/1522683421952y2z14-6602375682\
   199544287/1522683421952yz15-9904447712488549337/1522683421952z16-78458368\
   51981512509823/1622790636544y2z15-7845836851981512509823/1622790636544yz1\
   6-7845836851981512509823/1622790636544z17-5yz17-95/12z18-2296555/94yz18+1\
   2055805/564z19-1200204388940305/2141422272yz19-726023508016505/713807424z\
   20-196100702405618841515/2284025132928yz20-4393431116392277120081 5/173585\
   910102528z21-1304323830944394846862228 5/468986493961216yz21-1445817020185\
   13382166954765/1875945975844864z22+3099566692215409928555825/284761575014\
   4yz22-11814070788738534139430304881 3025/285143788328419328z23+10145568057\
   625313602254417069246 6625/192472057121683 0464z24
↦  j[100]=xy5z5+y6z5+y5z6+y10z3-55/6y8z5-88/3y7z6-154/3y6z7-176/3y5z8-275/6y\
   4z9-73/3y3z10-49/6y2z11-11/6yz12-1/3z13+26211/2y7z7+25817237/564y6z8+1477\
   6937/188y5z9+169837205/2068y4z10+7286321/132y3z11+48666813/2068y2z12+1427\
   3741/2068yz13+3596333/3102z14-1287510460349/62983008y6z9+23925416061685/4\
   61875392y4z11+262628419382125/2944455624y3z12+308026320020209/3925940832y\
   2z13+69773522330591/2617293888yz14+57164869308695/5888911248z15-196348286\
   37771265/9887554688y3z13+5978673815371681295/1522683421952y2z14+597866620\
   1954571535/1522683421952yz15+9002429812171346345/1522683421952z16+7138912\
   337581223050191/1622790636544y2z15+7138912337581223050191/1622790636544yz\
   16+7138912337581223050191/1622790636544z17+5yz17+95/12z18+2215235/94yz18-\
   11783635/564z19+1097712163576705/2141422272yz19+663249341441705/713807424\
   z20+1792947619908751028 75/2284025132928yz20+40016751223659310234975/17358\
   5910102528z21+118774979343713472434 62845/468986493961216yz21+131693580706\
   408643565147005/1875945975844864z22-2580890931269240863150 75/258874159104\
   yz22+1074774747971469833019808475630 25/285143788328419328z23-923146232529\
   154190319661212315910 25/192472057121683 0464z24
↦  j[101]=xy6z4+y7z4+y6z5-y10z3+55/6y8z5+88/3y7z6+154/3y6z7+176/3y5z8+275/6y\
   4z9+73/3y3z10+49/6y2z11+11/6yz12+1/3z13-24579/2y7z7-24221413/564y6z8-1386\
   6373/188y5z9-159359545/2068y4z10-6835309/132y3z11-45606837/2068y2z12-1336\
   2709/2068yz13-3357877/3102z14+1135380310349/62983008y6z9-21489137816485/4\
   61875392y4z11-237656058852445/2944455624y3z12-276145905102049/3925940832y\
   2z13-62325489107951/2617293888yz14-45064798965815/5888911248z15+173708533\
   32579585/9887554688y3z13-3988796004711358367/1522683421952y2z14-398879600\
   4711358367/1522683421952yz15-6663899804511504697/1522683421952z16-5432757\
   820012659581559/1622790636544y2z15-5432757820012659581559/1622790636544yz\
   16-5432757820012659581559/1622790636544z17-5yz17-95/12z18-1043420/47yz18+\
   11170415/564z19-953013569533105/2141422272yz19-553337624682905/713807424z\
   20-1458403191369339341 15/2284025132928yz20-30969892972452188138815/173585\
   910102528z21-92315949188691054097744 05/468986493961216yz21-10282907315529\
   7376672636245/1875945975844864z22+2512229266111330470845825/2847615750144\
   yz22-8144812184678210559619629015 8025/285143788328419328z23+7025294093232\
   630498309010395558 3025/192472057121683 0464z24
↦  j[102]=xy7z3+y8z3+y7z4+y10z3-55/6y8z5-88/3y7z6-154/3y6z7-176/3y5z8-275/6y\
   4z9-73/3y3z10-49/6y2z11-11/6yz12-1/3z13+20967/2y7z7+20668469/564y6z8+1183\
   6229/188y5z9+136042685/2068y4z10+5835617/132y3z11+38903001/2068y2z12+1137\
   8957/2068yz13+2838701/3102z14-949670063249/62983008y6z9+18982415596585/46\
   1875392y4z11+212247263980465/2944455624y3z12+238285193966389/3925940832y2\
   z13+51289774263611/2617293888yz14+21727413902435/5888911248z15-1323759792\
   507375/898868608y3z13+5045118569435207 5/1522683421952y2z14+50458799111461\
   835/1522683421952yz15+22929002742018453 25/1522683421952z16+23181027382997\
   63444907/1622790636544y2z15+23181027382997634449 07/1622790636544yz16+2318\
   102738299763444907/1622790636544z17+5yz17+95/12z18+1797545/94yz18-9586945\
   /564z19+67897253661455/194674752yz19+381479299986605/713807424z20+3194889\
   6001371716525/761341710976yz20+498753997953965166 2025/57861970034176z21+4\
   507422885797099806930065/468986493961216yz21+5166511852312241820392 2385/1\
   875945975844864z22-2106020805947952066448325/2847615750144yz22+3372837380\
   73889141116924625055 25/285143788328419328z23-29978277251273506173594 41936\
   9683925/192472057121683 0464z24
↦  j[103]=xy8z2+y9z2+y8z3-y10z3+55/6y8z5+88/3y7z6+154/3y6z7+176/3y5z8+275/6y\
   4z9+73/3y3z10+49/6y2z11+11/6yz12+1/3z13-14055/2y7z7-13844125/564y6z8-7930\
   585/188y5z9-91182325/2068y4z10-3914125/132y3z11-26077665/2068y2z12-759890\
   5/2068yz13-1846525/3102z14-56738190001/62983008y6z9-788583031735/46187539\
   2y4z11+3705043629665/2944455624y3z12+33698825888021/3925940832y2z13+26934\
   521105179/2617293888yz14+48443172337795/5888911248z15-167462684193408 5/98\
   87554688y3z13+6068824708329191147/1522683421952y2z14+6068817094912081387/\
   1522683421952yz15+6326709628569930477/1522683421952z16+438316603323713262\
   6075/1622790636544y2z15+4383166033237132626075/1622790636544yz16+43831660\
   33237132626075/1622790636544z17-5yz17-95/12z18-1228975/94yz18+6430475/564\
   z19+229433021403845/2141422272yz19+177561956970445/713807424z20+437251626\
   0828943805/761341710976yz20+6733889229863237423305/57861970034176z21+6472\
   489146665446781282625/468986493961216yz21+68736014372542198909702625/1875\
   945975844864z22-23546855007997122605792 5/2847615750144yz22+67598204349319\
   764879858861070725/285143788328419328z23-5667371275784443087250214799516 9\
   925/19247205712168 30464z24
↦ j[104]=xy9z+y10z+y9z2+y10z3-55/6y8z5-88/3y7z6-154/3y6z7-176/3y5z8-275/6y4\
   z9-73/3y3z10-49/6y2z11-11/6yz12-1/3z13+81/2y7z7+20987/564y6z8-29773/188y5\
   z9-937225/2068y4z10-76489/132y3z11-954237/2068y2z12-428269/2068yz13-24054\
   7/3102z14+1532345992013/31491504y6z9-28306817038645/230937696y4z11-334935\
   035606125/1472227812y3z12-414598563097033/1962970416y2z13-115993745512367\
   /1308646944yz14-95302123686515/2944455624z15+24638371580196585/4943777344\
   y3z13-7367766263566669687/761341710976y2z14-7367766263566669687/761341710\
   976yz15-1116207168020838889 7/761341710976z16-8866788100400929259007/81139\
   5318272y2z15-8866788100400929259007/811395318272yz16-886678810040092925 90\
   07/811395318272z17+5yz17+95/12z18-61520/47yz18-872485/564z19-144593453266\
   9585/1070711136yz19-802392457412585/356903712z20-51726760136722013505/380\
   670855488yz20-15717885863814697019405/28930985017088z21-14783421512249437\
   559345565/234493246980608yz21-16270053315437758830488 3885/937972987922432\
   z22+3226237691734140753 35275/129437079552yz22-1336268513535439798439829 88\
   039425/142571894164209664z23+11465182666555066980434032154767 8725/9623602\
   85608415232z24
↦ j[105]=y11+11/2y10z+35/2y9z2+75/2y8z3+45y7z4+21y6z5-21y5z6-45y4z7-75/2y3z\
   8-35/2y2z9-11/2yz10-z11-121/12y10z3+605/8y8z5+484/3y7z6+847/6y6z7-3025/24\
   y4z9-1639/12y3z10-605/8y2z11-605/24yz12-55/12z13-496557/8y7z7-163897893/7\
   52y6z8-283577679/752y5z9-3289198455/8272y4z10-47621529/176y3z11-973133811\
   /8272y2z12-289849887/8272yz13-25699797/4136z14+13790898641045/41988672y6z\
   9-254756053973725/307916928y4z11-2895489259664005/1962970416y3z12-3493588\
   014554305/2617293888y2z13-885441037185095/1744862592yz14-738852525441155/\
   3925940832z15+5879261940425482 5/1797737216y3z13-29029629870534918555/4350\
   52406272y2z14-29029639659214059675/435052406272yz15-43257433977685445085/\
   435052406272z16-48857654211614037 63189/66236352512y2z15-48857654211614037\
   63189/66236352512yz16-48857654211614037 63189/66236352512z17-1045/6yz17+32\
   45/48z18-24382485/188yz18+61143165/752z19-1125461947884875/129783168yz19-\
   7224008537147825/475871616z20-24268494197665747 1325/217526203136yz20-6220\
   846275054696464482 5/16531991438336z21-81138004309986888864 12255/191423058\
   75968yz21-89584020780427449954878895/76569223503872z22+531742344903106193\
   3045651675/324628195516416yz22-1988109506475858878586293866664425/3142400\
   93259890688z23+2339865429414017202634203238888925/2909630493147136z24
↦ j[106]=xy10-9/2y10z-35/2y9z2-75/2y8z3-45y7z4-21y6z5+21y5z6+45y4z7+75/2y3z\
   8+35/2y2z9+11/2yz10+z11+25/4y10z3-1045/24y8z5-308/3y7z6-231/2y6z7-176/3y5\
   z8+275/24y4z9+151/4y3z10+629/24y2z11+215/24yz12+7/4z13+602439/8y7z7+59631\
   7133/2256y6z8+344750573/752y5z9+4012256885/8272y4z10+175307009/528y3z11+1\
   204546857/8272y2z12+361624349/8272yz13+95745797/12408z14-44438750603353/1\
   25966016y6z9+820915339289345/923750784y4z11+8987440254211625/5888911248y3\
   z12+10571830073260373/7851881664y2z13+2396033126200627/5234587776yz14+203\
   7887782880815/11777822496z15-676897656616747605/19775109376y3z13+22235239\
   4201342493227/3045366843904y2z14+222352477948930700587/3045366843904yz15+\
   326594564799567636557/3045366843904z16+25713555278296054944 8547/324558127\
   3088y2z15+25713555278296054948547/3245581273088yz16+25713555278296054944\
   8547/3245581273088z17+165/2yz17-1015/48z18+31805095/188yz18-204052765/225\
   6z19+38065846621703885/4282844544yz19+23284988027782885/1427614848z20+634\
   5531001663158347815/4568050265856yz20+1435095300479986638034715/347171820\
   205056z21+4255897280268674662597018 65/937972987922432yz21+471184385628447\
   9645936490585/3751891951689728z22-55656138146794799078 69692925/3246281955\
   16416yz22+10463687571436195807672822693432 0975/1539776456973464 3712z23-33\
   249445788040618023355971957140172 25/3849441142433660928z24
↦ j[107]=xz9+yz9+z10-2y10z2-10y9z3-80/3y8z4-140/3y7z5-56y6z6-140/3y5z7-80/3\
   y4z8-10y3z9-8/3y2z10-2/3yz11+4765/8y10z3-20835/16y8z5+7895y7z6+130465/4y6\
   z7+114225/2y5z8+948125/16y4z9+314315/8y3z10+269835/16y2z11+78825/16yz12+6\
   155/8z13+10422463/144y7z7+30860986159/121824y6z8+11267616007/40608y5z9+24\
   51530645/40608y4z10-593889031/2592y3z11-3754807597/13536y2z12-5387471995/\
   40608yz13-1469311027/60912z14+1416414815/477144y6z9-8816857945/874764y4z1\
   1+6621221233 1860/11153241y3z12+530405865696055/59483952y2z13+177001804012\
   105/19827984yz14+531293352896875/178451856z15-4518128884213 48725/49437773\
   44y3z13-946795617184960 2645/69212882816y2z14-9467956171849602645/69212882\
   816yz15-3142575733950720495/69212882816z16+4669864624952322557/4149180604\
   8y2z15+4669865039870383037/41491806048yz16+4669864694105332637/4149180604\
   8z17+36475/4yz17-677825/32z18-7346744615/5076yz18-154285681835/121824z19+\
   777340330710 0775/178451856yz19-8644863356275/14870988z20-675417569920600 2\
   08925/34606441408yz20-118395500273292442575/12584160512z21+31401103304881\
   931520310995/202516895119616yz21+88742651433507504848607 0005/109359123364\
   59264z22-125175948580384691408467 8625/27052349626368yz22-4480654712428475\
   03622831835375/2916243289722470 4z23-708914987577177672282280241122 0525/47\
   24314129350402048z24
↦ j[108]=x2z8+2xyz8+y2z8-z10+4y10z2+20y9z3+160/3y8z4+280/3y7z5+112y6z6+280/\
   3y5z7+160/3y4z8+20y3z9+16/3y2z10+4/3yz11-4765/4y10z3+20835/8y8z5-15790y7z\
   6-130465/2y6z7-114225y5z8-948125/8y4z9-314315/4y3z10-269835/8y2z11-78825/\
   8yz12-6155/4z13-10422463/72y7z7-30860986159/60912y6z8-11267616007/20304y5\
   z9-2451530645/20304y4z10+593889031/1296y3z11+3754807597/6768y2z12+5387471\
   995/20304yz13+1469311027/30456z14-1399043615/238572y6z9+8715188485/437382\
   y4z11-132429370952180/11153241y3z12-530417086848295/29741976y2z13-1770032\
   39888425/9913992yz14-531293942874715/89225928z15+451831513300852725/24718\
   88672y3z13+1352622186881068995/4943777344y2z14+1352622186881068995/494377\
   7344yz15+448959160279363545/4943777344z16-666686281192956551/2963700432y2\
   z15-666686340466965191/2963700432yz16-666686291071957991/2963700432z17-36\
   475/2yz17+677825/16z18+7346744615/2538yz18+154285681835/60912z19-77735452\
   29804775/89225928yz19+8634602948425/7435494z20+964880821616 36295675/24718\
   88672yz20+18604956921683571 3075/9887554688z21-448604507060627989911285/1\
   4465492508544yz21-1267782488971393442225717 15/781136595461376z22+12518104\
   443971724502314786 25/13526174813184yz22+6403929609006890482381452 9625/208\
   3030921230336z23+1012192877731541557473812325964075/3374510092393144 32z24
↦ j[109]=xy2z7+y3z7+xyz8+2y2z8+yz9-20y10z3+135y8z5+100y7z6-280y6z7-750y5z8-\
   860y4z9-580y3z10-240y2z11-75yz12+5z13-2430y7z7-8505y6z8-9315y5z9-2025y4z1\
   0+7695y3z11+9315y2z12+4455yz13+810z14-137720/1269y6z9+1667195/4653y4z11-2\
   781943060/13959y3z12-1392968470/4653y2z13-464878555/1551yz14-1395517225/1\
   3959z15+59336141 1525/193358y3z13+12453489091035/2707012y2z14+12453489091 0\
   35/2707012yz15+4146429329685/2707012z16-22233021781185/24522344y2z15-2223\
   3021781185/24522344yz16-22233021781185/24522344z17-25yz17+775z18+48600yz1\
   8+42525z19-20413773700/13959yz19+183517925/9306z20+443546916219075/676753\
   yz20+1711538620047975/5414024z21-107889802199840092 5/208439924yz21-274335\
   714970145775/104219962z22+1652695466624865838625/1058054976yz22+834251599\
   282706046264875/1140583264128z23+7746570213321534310029375/506925895168z2\
   4
↦ j[110]=x2yz7-y3z7-2y2z8-yz9+40y10z3-270y8z5-200y7z6+560y6z7+1500y5z8+1720\
   y4z9+1160y3z10+480y2z11+150yz12-10z13+4860y7z7+17010y6z8+18630y5z9+4050y4\
   z10-15390y3z11-18630y2z12-8910yz13-1620z14+229240/1269y6z9-2793595/4653y4\
   z11+94640854120/237303y3z12+47376934720/79101y2z13+15808141945/26367yz14+\
   47449455190/237303z15-20187642625875/3287086y3z13-423704404766295/4601920\
   4y2z14-423704404766295/46019204yz15-141077408004045/46019204z16+426571448\
   61645/24522344y2z15+42657144861645/24522344yz16+42657144861645/24522344z1\
   7+50yz17-1550z18-97200yz18-85050z19+694266994450/237303yz19-3066280300/79\
   101z20-15080112577645275/11504801yz20-58189264261463325/92038408z21+14675\
   7347288825452425/14173914832yz21+1492360321682564867 25/28347829664z22-112\
   614287630425887442375/35973869184yz22-570958146707885458301961 25/38779830\
   980352z23-128226219869267477322323625/4308870108928z24
↦ j[111]=xy3z6+y4z6+y3z7-xyz8-y2z8-yz9+30y10z3-405/2y8z5-306y7z6-126y6z7+16\
   2y5z8+495/2y4z9+144y3z10+81/2y2z11+57/2yz12-18z13+6561/2y7z7+45927/4y6z8+\
   50301/4y5z9+10935/4y4z10-41553/4y3z11-50301/4y2z12-24057/4yz13-2187/2z14+\
   3454/47y6z9-121350/517y4z11+139340435/517y3z12+418321803/1034y2z13+418559\
   333/1034yz14+139570115/1034z15-3212747506665/773432y3z13-6133920682635/98\
   4368y2z14-6133920682635/984368yz15-22494662415675/10828048z16+78018150537\
   27/49044688y2z15+7801815053727/49044688yz16+7801815053727/49044688z17-405\
   yz17-4455/4z18-65610yz18-229635/4z19+1021492805/517yz19-25863195/1034z20-\
   4790677244384925/5414024yz20-9244453123161225/21656096z21+585329039277152\
   7645/833759696yz21+5878965984667136415/1667519392z22-245468518138517297 17\
   5/117561664yz22-692567449115946674 6175/6670077568z23-549896891331543554 48\
   82975/1013851790336z24
↦ j[112]=xy4z5+y5z5+y4z6+xyz8+y2z8+yz9-50y10z3+270y8z5+292y7z6-238y6z7-894y\
   5z8-1010y4z9-628y3z10-231y2z11-87yz12+11z13-5346y7z7-18711y6z8-20493y5z9-\
   4455y4z10+16929y3z11+20493y2z12+9801yz13+1782z14-107426/1269y6z9+1238705/\
   4653y4z11-104279949410/237303y3z12-52166850719/79101y2z13-17395705898/263\
   67yz14-52200514085/237303z15+22263444302265/3287086y3z13+467423176131315/\
   46019204y2z14+467423176131315/46019204yz15+155734955899605/46019204z16-93\
   56490298143/6130586y2z15-9356490298143/6130586yz16-9356490298143/6130586z\
   17+485yz17+1840z18+106920yz18+93555z19-764254156340/237303yz19+3168673535\
   /79101z20+16589299116393375/11504801yz20+64014825565742775/92038408z21-16\
   245137386890892660 5/14173914832yz21-20514628345434412995/3543478708z22+12\
   230408071497023421972 5/35973869184yz22+3099966098407440503945807 5/1938991\
   5490176z23+21155378198368342478119357 5/8617740217856z24
↦ j[113]=xy5z4+y6z4+y5z5-xyz8-y2z8-yz9+75y10z3-675/2y8z5-498y7z6-168y6z7+30\
   6y5z8+795/2y4z9+192y3z10+63/2y2z11+51/2yz12-24z13+12393/2y7z7+86751/4y6z8\
   +95013/4y5z9+20655/4y4z10-78489/4y3z11-95013/4y2z12-45441/4yz13-4131/2z14\
   +902/47y6z9-20940/517y4z11+4481086570/8789y3z12+13444813493/17578y2z13+13\
   445349673/17578yz14+4482215115/17578z15-854678163465/108664y3z13-21714741\
   66899745/184076816y2z14-217147416689974 5/184076816yz15-723649357990035/18\
   4076816z16+106352070291801/49044688y2z15+106352070291801/49044688yz16+106\
   352070291801/49044688z17-915yz17-8715/4z18-123930yz18-433755/4z19+5967905\
   885/1598yz19-71510445/1598z20-153838794976230825/92038408yz20-29680965044\
   8392825/368153632z21+379100754580207174965/28347829664yz21+19102901522990\
   0486715/28347829664z22-846508515373121364347 5/2180234496yz22-228639664680\
   14959972956575/12926610326784z23-514360704677516824873740975/172354804357\
   12z24
↦  j[114]=xy6z3+y7z3+y6z4+xyz8+y2z8+yz9-355/2y10z3+4095/4y8z5+1804y7z6+959y6\
   z7-1038y5z8-8765/4y4z9-3587/2y3z10-3363/4y2z11-1161/4yz12-41/2z13-58563/4\
   y7z7-409941/8y6z8-448983/8y5z9-97605/8y4z10+370899/8y3z11+448983/8y2z12+2\
   14731/8yz13+19521/4z14-456709/2538y6z9+2648150/4653y4z11-16803562855/1395\
   9y3z12-33620602921/18612y2z13-1019124317/564yz14-33637230565/55836z15+287\
   15244673095/1546864y3z13+602655259692735/21656096y2z14+602655259692735/21\
   656096yz15+200641834269405/21656096z16-769409234887941/98089376y2z15-7694\
   09234887941/98089376yz16-769409234887941/98089376z17-505yz17+27365/8z18+2\
   92815yz18+2049705/8z19-492544730935/55836yz19+1014561545/9306z20+42759769\
   181783475/10828048yz20+82486202679399975/43312192z21-10548438775411220419\
   5/3335038784yz21-53480743988955730695/3335038784z22+775647419365109709660\
   25/8464439808yz22+18278145748430711894029175/4562333056512z23+22714341902\
   5045581602456175/2027703580672z24
↦  j[115]=xy7z2+y8z2+y7z3-xyz8-y2z8-yz9+1575/4y10z3-21105/8y8z5-5310y7z6-850\
   5/2y6z7+450y5z8+33255/8y4z9+16605/4y3z10+17505/8y2z11+6075/8yz12+405/4z13\
   +295245/8y7z7+2066715/16y6z8+2263545/16y5z9+492075/16y4z10-1869885/16y3z1\
   1-2263545/16y2z12-1082565/16yz13-98415/8z14+292655/564y6z9-1745765/1034y4\
   z11+79997752370/26367y3z12+320135288395/70312y2z13+320254000415/70312yz14\
   +320314393255/210936z15-2460867528553425/52593376y3z13-4695016674413115/6\
   6937024y2z14-4695016674413115/66937024yz15-17193038018796315/736307264z16\
   +3522639307208265/196178752y2z15+3522639307208265/196178752yz16+352263930\
   7208265/196178752z17+7425/2yz17-81675/16z18-1476225/2yz18-10333575/16z19+\
   4690224057275/210936yz19-14594633575/52734z20-3664755290597206725/3681536\
   32yz20-7069437419650553025/1472614528z21+9050586201699896526975/113391318\
   656yz21+2289566926700140050675/56695659328z22-737379565608980793653125/31\
   976772608yz22-87699613731545348385773125/8617740217856z23-182612406319939\
   24170922643625/68941921742848z24
↦  j[116]=xy8z+y9z+y8z2+xyz8+y2z8+yz9-5335/8y10z3+59625/16y8z5+7111y7z6+2066\
   9/4y6z7-2859/2y5z8-99575/16y4z9-47057/8y3z10-50433/16y2z11-17211/16yz12-1\
   121/8z13-1086453/16y7z7-7605171/32y6z8-8329473/32y5z9-1810755/32y4z10+688\
   0869/32y3z11+8329473/32y2z12+3983661/32yz13+362151/16z14-7832869/10152y6z\
   9+94544155/37224y4z11-1324972802525/237303y3z12-10603655889767/1265616y2z\
   13-3535623463679/421872yz14-10608492320135/3796848z15+9061585278322365/10\
   5186752y3z13+190169117118073845/1472614528y2z14+190169117118073845/147261\
   4528yz15+63306923221560735/1472614528z16-15268309186330971/392357504y2z15\
   -15268309186330971/392357504yz16-15268309186330971/392357504z17-13105/4yz\
   17+242915/32z18+5432265/4yz18+38025855/32z19-155350412628845/3796848yz19+\
   319673598265/632808z20+13486659696913514325/736307264yz20+236504046359113\
   0575/267748096z21-33406180995035979489015/226782637312yz21-16896023182121\
   198479515/226782637312z22+24251267359343224416321425/575581906944yz22+292\
   591298284635838904393275/16328349886464z23+73556186438621967928969329225/\
   137883843485696z24
↦  j[117]=xy9+y10+y9z+y10z2+5/3y9z3-5/3y8z4-50/3y7z5-42y6z6-182/3y5z7-170/3y\
   4z8-35y3z9-41/3y2z10-11/3yz11-2/3z12+4765/8y10z3-20835/16y8z5+7895y7z6+13\
   0465/4y6z7+114225/2y5z8+948125/16y4z9+314315/8y3z10+269835/16y2z11+78825/\
   16yz12+6155/8z13+10426687/144y7z7+30873054127/121824y6z8+11274323719/4060\
   8y5z9+2458368725/40608y4z10-593601799/2592y3z11-3754171309/13536y2z12-538\
   6884667/40608yz13-1469170099/60912z14+157775815/59643y6z9-32234694805/349\
   9056y4z11+132459543842675/22306482y3z12+530472611040205/59483952y2z13+885\
   03433475315/9913992yz14+531270309400015/178451856z15-41040285445422225/44\
   9434304y3z13-9462830112598910535/69212882816y2z14-9462830112598910535/692\
   12882816yz15-3142626154003887885/69212882816z16+29296924739250320455/3319\
   34448384y2z15+29296922526353997895/331934448384yz16+29296929165042965575/\
   331934448384z17+36475/4yz17-677825/32z18-7346578175/5076yz18-154292180075\
   /121824z19+88347631988125/2027862yz19-3117406501025/5407632z20-6752285131\
   16377969275/34606441408yz20-130215171004877887597 5/138425765632z21+139228\
   5199192582215996876 25/911326028038272yz21+174396639931163320301189537 5/21\
   871824672918528z22-11850846919060730046870762 5/2459304511488yz22-10760550\
   462114275570736659521 25/58324865794449408z23-583823176855839606001217152 2\
   566425/4724314129350402048z24
↦ j[118]=x2y8-y10-2y9z-y8z2-2xyz8-2y2z8-2yz9-2y10z2-10/3y9z3+10/3y8z4+100/3\
   y7z5+84y6z6+364/3y5z7+340/3y4z8+70y3z9+82/3y2z10+22/3yz11+4/3z12+285/2y10\
   z3-19395/4y8z5-30012y7z6-75567y6z7-111366y5z8-424275/4y4z9-133629/2y3z10-\
   109701/4y2z11-30807/4yz12-2517/2z13-324305/36y7z7-960084065/30456y6z8-352\
   111241/10152y5z9-80260315/10152y4z10+18125705/648y3z11+115402115/3384y2z1\
   2+165809429/10152yz13+45230621/15228z14-100544483/26508y6z9+237709835/176\
   72y4z11-878887041815/1239249y3z12-445776746493/413083y2z13-3610665711089/\
   3304664yz14-1814970972335/4956996z15+6371084862201555/617972168y3z13+6551\
   6470444195875/4325805176y2z14+65516470444195875/4325805176yz15+1045943820\
   4392495/2162902588z16-1322346584403667349/15087929472y2z15-13223463832312\
   74389/15087929472yz16-1322346986748453269/15087929472z17-11685yz17+217455\
   /8z18+226516945/1269yz18+4763875045/30456z19-17518941026505/3304664yz19+1\
   433940976805/9913992z20+10335055322881190925/4325805176yz20+9929241502658\
   605125/8651610352z21-86464279047274863233099 5/82847820730752yz21-10026096\
   865611471994802315/994173848769024z22+70371757259632902604844 75/563590617\
   216yz22+1956021496085718380059557275/883710087794688z23+12534642939968108\
   162393838568625/9760979606095872z24
↦ j[119]=x2z7+2xyz7+y2z7+2xz8+2yz8+z9+30800/423y6z8+30800/141y5z9+486470/15\
   51y4z10+26060/99y3z11+70670/517y2z12+64340/1551yz13+88810/4653z14-7405525\
   /24299y6z9+353409675/534578y4z11+5600660600/4543913y3z12+5809184925/45439\
   13y2z13+5610405375/9087826yz14+1593357500/4543913z15-9744523043625/308986\
   084y3z13+7865902302069825/47583856936y2z14+7865902302069825/47583856936yz\
   15+9366558850788075/47583856936z16+2146620481075060 95/1584756481y2z15+214\
   662048107506095/1584756481yz16+214662048107506095/1584756481z17+49000/141\
   yz18+91175/423z19+8122630625/826166yz19+1475494125/75106z20+1556636160857\
   9625/11895964234yz20+612200237651841375/95167713872z21+535252907767487420\
   2025/7327913968144yz21+8710770494389825722302 5/43967483808864z22-15588448\
   3661694095625/9887554688yz22+13791795622341152135180456 25/117246623490304\
   z23-266465097028654098023229668060875/180442553551577856z24
↦ j[120]=xy2z6+y3z6+xyz7+2y2z7+yz8-20y10z2-100y9z3-315y8z4-660y7z5-840y6z6-\
   630y5z7-210y4z8+60y3z9+90y2z10+35yz11+25z12-1852/3y7z7-2785226/1269y6z8-1\
   617746/423y5z9-18848465/4653y4z10-818198/297y3z11-1842938/1551y2z12-16031\
   03/4653yz13-949063/13959z14-200660825/72897y6z9+2647176050/267289y4z11+24\
   7943744230/13631739y3z12+79951936540/4543913y2z13+34949943690/4543913yz14\
   +29435639665/13631739z15-750364472331675/308986084y3z13-190201051266966 67\
   5/47583856936y2z14-19020105126696667 5/47583856936yz15-746449225278887 25/4\
   7583856936z16-1145225780365303463 9/50712207392y2z15-1145222357291304503 9/\
   50712207392yz16-1145223498315970823 9/50712207392z17-563860/423yz18+200328\
   5/2538z19+161736531775/2478498yz19+43666153975/413083z20-1080912810452488\
   875/11895964234yz20-4355066495750932125/95167713872z21+18610162635696969 7\
   55887045/417691096184208yz21+234245080384114055828470205/1002458630842099\
   2z22+767027925222825847661875/19775109376yz22-1815000888176284591863541 87\
   5/468986493961216z23+666402566762256623805660342503825/270663830327366784\
   z24
```

```
↦ j[121]=x2yz6-y3z6-2y2z7-yz8+40y10z2+200y9z3+630y8z4+1320y7z5+1680y6z6+126\
  0y5z7+420y4z8-120y3z9-180y2z10-70yz11-50z12+3392/3y7z7+5062336/1269y6z8+2\
  917726/423y5z9+33733390/4653y4z10+1453558/297y3z11+3249508/1551y2z12+2849\
  128/4653yz13+1667198/13959z14+824859875/145794y6z9-21530818075/1069156y4z\
  11-503166594530/13631739y3z12-324495046255/9087826y2z13-282966185235/1817\
  5652yz14-60700373165/13631739z15+273674327539725/56179288y3z13+1075367159\
  69527575/13595387696y2z14+107536715969527575/13595387696yz15+413075287049\
  14125/13595387696z16+1390709606701412201/3622300528y2z15+1390704716595699\
  401/3622300528yz16+1390706346630937001/3622300528z17+929435/423yz18-20988\
  05/1269z19-60756900425/450636yz19-365571595775/1652332z20+153631004775148\
  125/849711731yz20+2397615030608356125/27190775392z21-10677692774577796473\
  522745/119340313195488yz21-8543001261389272550615555/179010469793232z22-1\
  533912321925569677241875/19775109376yz22+15562075862209915374841875/83747\
  58820736z23-647400216245483540020440549874525/154665045901352448z24
↦ j[122]=xy3z5+y4z5+y3z6-xyz7-y2z7-yz8+30y10z2+135y9z3+405y8z4+720y7z5+630y\
  6z6-630y4z8-720y3z9-405y2z10-120yz11-45z12+2078y7z7+340566/47y6z8+583893/\
  47y5z9+6709595/517y4z10+95958/11y3z11+1933377/517y2z12+569694/517yz13+102\
  533/517z14-94912875/97196y6z9-2911102875/2138312y4z11-13490258445/4543913\
  y3z12-178197534495/18175652y2z13-306906320115/36351304yz14-53243896545/90\
  87826z15+5204006726201775/1235944336y3z13+171211027298041575/27190775392y\
  2z14+171211027298041575/27190775392yz15+56722879321602525/27190775392z16-\
  12628824906906423/1034943008y2z15-12630222079967223/1034943008yz16-126295\
  23493436823/1034943008z17+341835/94yz18-290635/94z19+13423502475/3304664y\
  z19-314243831475/3304664z20+320302952555061375/1699423462yz20+42384348010\
  01854125/54381550784z21-1026776842710772806212955/11365744113856yz21-5814\
  88683445294414836705/11365744113856z22-3080895255663027470128125/39550218\
  752yz22-164562449972300743426396875/4785576468992z23+21707136779621806841\
  0739043975/1636667152395264z24
↦ j[123]=xy4z4+y5z4+y4z5+xyz7+y2z7+yz8-50y10z2-220y9z3-720y8z4-1380y7z5-147\
  0y6z6-630y5z7+420y4z8+780y3z9+495y2z10+155yz11+55z12-9736/3y7z7-14313938/\
  1269y6z8-8159288/423y5z9-93567545/4653y4z10-4008014/297y3z11-8977919/1551\
  y2z12-7971314/4653yz13-4227799/13959z14+68457475/291588y6z9+13681240225/2\
  138312y4z11+177807853615/13631739y3z12+366210664405/18175652y2z13+4998402\
  44985/36351304yz14+187299515465/27263478z15-724354328812725/112358576y3z1\
  3-2076609628894013775/190335427744y2z14-2076609628894013775/190335427744y\
  z15-849553395885257625/190335427744z16-160256670164141259683/202848829568\
  y2z15-160256259395261384483/202848829568yz16-160256441959207995683/202848\
  829568z17-2308780/423yz18+6216265/1269z19+8685932675/901272yz19+322019002\
  025/3304664z20-27489769156849798125/95167713872yz20-58343927494571985375/\
  380670855488z21+440042360916873913967311355/3341528769473664yz21+13866977\
  9041579387287118415/2110439222825472z22+4618755900045255937019375/3955021\
  8752yz22-36008438343336428263980999375/1875945975844864z23+18651077950560\
  763379965724423502925/2165310642618934272z24
↦ j[124]=xy5z3+y6z3+y5z4-xyz7-y2z7-yz8+75y10z2+255y9z3+810y8z4+1440y7z5+126\
  0y6z6-1260y4z8-1440y3z9-810y2z10-255yz11-75z12+5806y7z7+2841226/141y6z8+1\
  616641/47y5z9+18500915/517y4z10+790633/33y3z11+5298074/517y2z12+1573338/5\
  17yz13+822383/1551z14-1051504275/194392y6z9+18635231025/4276624y4z11+2532\
  457335/534578y3z12-19479647115/2138312y2z13-57594525255/4276624yz14-90214\
  64715/1069156z15+1178008818946425/145405216y3z13+334460480269289625/22392\
  403264y2z14+334460480269289625/22392403264yz15+153047122151540175/2239240\
  3264z16+719571432155612807313/405697659136y2z15+719570336771933140113/405\
  697659136yz16+719570884463772973713/405697659136z17+444335/47yz18-2577985\
  /282z19+41330722575/388784yz19+21221903925/388784z20+4500217777107531375/\
   11196201632yz20+10959190540020356625/44784806528z21-22427829967440707567\
   27655/131040343900928yz21-40042442452331792855977495/524161375603712z22-6\
   596921687997118619812 5/422996992yz22+19349233550602257185524153125/220699\
   526569984z23-5473494749552854060682863437989 75/28304714282600448z24
↦ j[125]=xy6z2+y7z2+y6z3+xyz7+y2z7+yz8-355/2y10z2-1205/2y9z3-3375/2y8z4-277\
   5y7z5-2415y6z6-315y5z7+1725y4z8+4125/2y3z9+2295/2y2z10+745/2yz11+100z12-3\
   1150/3y7z7-45777290/1269y6z8-26050685/423y5z9-297999890/4653y4z10-1272351\
   5/297y3z11-28342040/1551y2z12-25246850/4653yz13-13125775/13959z14+2663907\
   5875/1166352y6z9-385849344275/8553248y4z11-4315826223875/54526956y3z12-38\
   48424154775/72702608y2z13-1137409456875/145405216yz14+368262621275/109053\
   912z15-526813655107578 75/4943777344y3z13-18532123461292345875/76134171097\
   6y2z14-18532123461292345875/761341710976yz15-10419193172635633125/7613417\
   10976z16-43201219334172715256 75/811395318272y2z15-43201189211121524408 75/\
   811395318272yz16-43201207467516185528 75/811395318272z17-14272175/846yz18+\
   42741875/2538z19-22279985045375/39655968yz19-9601551844375/13218656z20-24\
   71415185022912281 25/380670855488yz20-77222342064514105687 5/1522683421952z\
   21+32135539750840188266204633 75/13366115077894656yz21+11326527703831819 81\
   3911045125/16039338093473587 2z22+37056842393615457279160625/158200875008y\
   z22-26935199505128702003686054443 75/7503783903379456z23+50276868210273583\
   14166350501514671 25/86612425704757370 88z24
↦ j[126]=xy7z+y8z+y7z2-xyz7-y2z7-yz8+1575/4y10z2+6075/4y9z3+16785/4y8z4+124\
   65/2y7z5+8505/2y6z6-2745/2y5z7-5715y4z8-22275/4y3z9-11295/4y2z10-3645/4yz\
   11-405/2z12+26386y7z7+25870777/282y6z8+7363186/47y5z9+84211265/517y4z10+7\
   190251/66y3z11+23985749/517y2z12+7127968/517yz13+3682213/1551z14-70261363\
   275/777568y6z9+3578674608225/17106496y4z11+13803120524955/36351304y3z12+4\
   2559793289405/145405216y2z13+24282118238985/290810432yz14+243093061005/72\
   702608z15+21982905366987 8025/9887554688y3z13+81935593458328091025/1522683\
   421952y2z14+81935593458328091025/1522683421952yz15+48081919193166875175/1\
   522683421952z16+21153454501046869733877/1622790636544y2z15+21153443273364\
   153145077/1622790636544yz16+21153454227200949817077/1622790636544z17+4069\
   895/94yz18-12315035/282z19+59104279557975/26437312yz19+6914963996775/2403\
   392z20+12720154145636728338 75/761341710976yz20+39489997467680983226 25/304\
   5366843904z21-509197256217435227528493091 5/8910743385263104yz21-523404462\
   633522422353622603 5/35642973541052416z22-17880942762695025525579187 5/3164\
   01750016yz22+12999472603018820168300985425625/15007567806758912z23-273534\
   50842954176497825487318130617 5/19247205712168304 64z24
↦ j[127]=x2y7+2xy8+y9-y7z2+2xyz7+2y2z7+2yz8-1575/2y10z2-6075/2y9z3-16785/2y\
   8z4-12465y7z5-8505y6z6+2745y5z7+11430y4z8+22275/2y3z9+11295/2y2z10+3645/2\
   yz11+405z12-57430y7z7-84523900/423y6z8-48159295/141y5z9-551327365/1551y4z\
   10-23566060/99y3z11-52476815/517y2z12-46781395/1551yz13-24180350/4653z14+\
   41319886975/194392y6z9-2143671977625/4276624y4z11-8272270195775/9087826y3\
   z12-26190379979925/36351304y2z13-15938336340225/72702608yz14-414059030575\
   /18175652z15-1019808208657541 25/2471888672y3z13-42412245122448887625/3806\
   70855488y2z14-42412245122448887625/380670855488yz15-26707198709122752375/\
   380670855488z16-12801799511653343325675/405697659136y2z15-12801793897811 9\
   85031275/405697659136yz16-12801799374730383367275/405697659136z17-1356170\
   0/141yz18+39724100/423z19-35030403243875/6609328yz19-46842320240625/66093\
   28z20-65252115180286612312 5/190335427744yz20-21911832300665782181 25/76134\
   1710976z21+24728529657673451741035321 25/2227685846315776yz21+541274186016\
   885269965816737 5/2673223015578931 2z22+89531761268106855553048125/79100437\
   504yz22-82511045070346471684767504693 75/3751891951689728z23+2613773009414\
   86711909017105224638 75/7597581202171699 2z24
↦ j[128]=x2z6+2xyz6+y2z6+2xz7+2yz7+z8-40/3y10z3+90y8z5+200/3y7z6-560/3y6z7-\
   500y5z8-1720/3y4z9-1160/3y3z10-160y2z11-50yz12+10/3z13-1620y7z7-5670y6z8-\
   6210y5z9-1350y4z10+5130y3z11+6210y2z12+2970yz13+540z14-225940/3807y6z9+25\
   2800/1269y4z11-94644147700/711909y3z12-47377768060/237303y2z13-1580834722\
   0/79101yz14-47446905700/711909z15+3364916372775/1643543y3z13+706265121681\
   45/23009602y2z14+70626512168145/23009602yz15+23517682949295/23009602z16-5\
   65561885095/1114652y2z15-565561885095/1114652yz16-565561885095/1114652z17\
   -50/3yz17+1550/3z18+32400yz18+28350z19-694290436600/711909yz19+3070849900\
   /237303z20+5026652081876800/11504801yz20+9698213345429825/46019204z21-222\
   2587172966312925/644268856yz21-2259131759526082725/1288537712z22+56391180\
   289000559885375/53960803776yz22+2633081850556857806438875/5288158770048z2\
   3+357622827165506138337875/391715464448z24
↦ j[129]=xy2z5+y3z5+xyz6+2y2z6+yz7-20y10z-100y9z2-315y8z3-660y7z4-840y6z5-6\
   30y5z6-210y4z7+60y3z8+90y2z9+35yz10+25z11-395/9y10z3+735/2y8z5-1016/9y7z6\
   -22456/9y6z7-16576/3y5z8-115975/18y4z9-41521/9y3z10-12373/6y2z11-3721/6yz\
   12-1048/9z13+1869669/2y7z7+614619501/188y6z8+1062924363/188y5z9+123356141\
   55/2068y4z10+179297493/44y3z11+3687230007/2068y2z12+1110546099/2068yz13+9\
   5800659/1034z14+804397012860647/188949024y6z9-15602820876389455/138562617\
   6y4z11-203853103793475175/8833366872y3z12-259235050033539427/11777822496y\
   2z13-84407381722819373/7851881664yz14-56357488165829285/17666733744z15+46\
   314929244815159 85/9887554688y3z13-4093246955683945 21695/1522683421952y2z1\
   4-409323667757084704095/1522683421952yz15-112257392073100810 4985/15226834\
   21952z16-100431024226985794 2646671/1622790636544y2z15-100431024226985794 2\
   646671/1622790636544yz16-1004310242269857942646671/1622790636544z17-24355\
   /9yz17+57595/36z18+193829265/94yz18-223553505/188z19-818412352222464115/6\
   424266816yz19-32803307856490465/194674752z20-5388710239243891036475/22840\
   25132928yz20-263107757176333952120725/9136100531712z21-182244705469419476\
   2245764445/468986493961216yz21-20080245422183401652789889405/187594597584\
   4864z22+113986257570215211655744715675/486942293274624yz22-12088489379792\
   710987377229557509 21825/2309664685460196 5568z23+144288002941420031721582 0\
   729865963225/213857841246314496z24
↦ j[130]=x2yz5-y3z5-2y2z6-yz7+40y10z+200y9z2+630y8z3+1320y7z4+1680y6z5+1260\
   y5z6+420y4z7-120y3z8-180y2z9-70yz10-50z11+850/9y10z3-780y8z5+796/9y7z6+42\
   476/9y6z7+31976/3y5z8+112300/9y4z9+80426/9y3z10+11974/3y2z11+3628/3yz12+2\
   018/9z13-1869102y7z7-307216479/47y6z8-531360027/47y5z9-6167562795/517y4z1\
   0-89668497/11y3z11-1844738703/517y2z12-555810471/517yz13-95898372/517z14-\
   804398851081031/94474512y6z9+15602866124891215/692813088y4z11+20405975465\
   4691975/4416683436y3z12+259647898634648371/5888911248y2z13+84682357715382\
   029/3925940832yz14+56563631218665005/8833366872z15-463506692185054390 5/49\
   43777344y3z13+5835675922066494946 5/108763101568y2z14+583566123904778326 65\
   /108763101568yz15+160328133614585504175/108763101568z16+14347281441368576\
   1777849/115913616896y2z15+143472814413685761777849/115913616896yz16+14347\
   281441368576177784 9/115913616896z17+46130/9yz17-30830/9z18-194362245/47yz\
   18+111310395/47z19+819511598522525395/3212133408yz19+3608324408508363 95/1\
   070711136z20+744860731757269593725/163144652352yz20+375386403523201080850\
   75/652578609408z21+260390498778722697021188955/33499035282944yz21+2868685\
   511574385377758919195/133996141131776z22-11407304558609039694107954727 5/2\
   43471146637312yz22+9089070449293757356070411311657325/8682949945339084 8z2\
   3-2061255899835037613131995147707347 75/15275560089022464z24
↦ j[131]=xy3z4+y4z4+y3z5-xyz6-y2z6-yz7+30y10z+135y9z2+405y8z3+720y7z4+630y6\
   z5-630y4z7-720y3z8-405y2z9-120yz10-45z11+120y10z3-2295/2y8z5+128y7z6+6958\
   y6z7+15744y5z8+36925/2y4z9+13243y3z10+11847/2y2z11+3609/2yz12+324z13-3722\
   193/2y7z7-1224482697/188y6z8-2122324011/188y5z9-24701770935/2068y4z10-360\
   682821/44y3z11-7470910179/2068y2z12-2265955803/2068yz13-197536023/1034z14\
   -5532300820049/874764y6z9+109077511802485/6414936y4z11+1491461921083975/4\
   0895217y3z12+1927375726721059/54526956y2z13+666811250933291/36351304yz14+\
   212704937294110/40895217z15-8911924564648224l5/1235944336y3z13+1702389304\
   3708962405/190335427744y2z14+17023636090881508005/190335427744yz15+154267\
   402862877887115/190335427744z16+151327952333723984400639/202848829568y2z1\
   5+151327952333723984400639/202848829568yz16+151327952333723984400639/2028\
   48829568z17+7740yz17-19785/4z18-202942665/47yz18+411812235/188z19+5891205\
   044832205/29741976yz19+2337511888271705/9913992z20-30608220886131146l825/\
   95167713872yz20+23341500055902466466557 5/7232746254272z21+290142075266988\
   130032299505/58623311745152yz21+3191458641469791325291722645/234493246980\
   608z22-2437033022855320512496464425/6763087406592yz22+2004887871272822271\
   1957037860303825/320786761869471744z23-3623478771502765528379800187878702\
   5/4455371692631552z24
↦ j[132]=xy4z3+y5z3+y4z4+xyz6+y2z6+yz7-50y10z-220y9z2-720y8z3-1380y7z4-1470\
   y6z5-630y5z6+420y4z7+780y3z8+495y2z9+155yz10+55z11-2185/18y10z3+5235/4y8z\
   5-6128/9y7z6-88543/9y6z7-63808/3y5z8-884225/36y4z9-314711/18y3z10-93413/1\
   2y2z11-28451/12yz12-7613/18z13+11192229/4y7z7+78319203/8y6z8+135607689/8y\
   5z9+1576214865/88y4z10+1079368113/88y3z11+474097221/88y2z12+143337897/88y\
   z13+12451227/44z14+42540018227713/4020192y6z9-833270903243945/29481408y4z\
   11-1115883394220502 5/187943976y3z12-14307641434744733/250591968y2z13-4816\
   559171447467/167061312yz14-3121285971998215/375887952z15+2496681094159812\
   15/210373504y3z13-16767888959987826l5/4628217088y2z14-1676779523859179415\
   /4628217088yz15-7169482096406145345/4628217088z16-673231867258653422516 7/\
   4932494336y2z15-6732318672586534225167/4932494336yz16-6732318672586534225\
   167/4932494336z17-89740/9yz17+460745/72z18+12674835/2yz18-27334515/8z19-4\
   4312417754736085/136686528yz19-18420711874503085/45562176z20+196812641159\
   8600925/6942325632yz20-323678190894158968871 75/527616748032z21-1258795595\
   2956102969534765/1425490863104yz21-13862483512323517002296668 5/5701963452\
   416z22+6144605253204459857104248325/10360474324992yz22-806195415128807593\
   6198066187361025/70202574026145792z23+96721802513568798482181389179l8325/\
   650023833575424z24
↦ j[133]=xy5z2+y6z2+y5z3-xyz6-y2z6-yz7+75y10z+255y9z2+810y8z3+1440y7z4+1260\
   y6z5-1260y4z7-1440y3z8-810y2z9-255yz10-75z11+205/4y10z3-7635/8y8z5+3116y7\
   z6+32837/2y6z7+31488y5z8+277525/8y4z9+96239/4y3z10+84091/8y2z11+25417/8yz\
   12+2247/4z13-29902689/8y7z7-9837034281/752y6z8-17023686003/752y5z9-197722\
   000755/8272y4z10-2876744133/176y3z11-59271248367/8272y2z12-17890413219/82\
   72yz13-1558721529/4136z14-11064857071703/874764y6z9+109080655576085/32074\
   68y4z11+2930351142316300/40895217y3z12+7499086179696371/109053912y2z13+12\
   63447963369827/36351304yz14+3192705945375635/327161736z15-351579544177336\
   6185/2471888672y3z13+7940152453493596288 5/380670855488y2z14+7940049672362\
   6145285/380670855488yz15+62083350866237944657 5/380670855488z16+1513275320\
   0332223344665 9/101424414784y2z15+151327532003322233446659/101424414784yz1\
   6+151327532003322233446659/101424414784z17+24685/2yz17-140105/16z18-78236\
   1585/94yz18+3500361405/752z19+11502139132628635/29741976yz19+233810206494\
   0505/4956996z20-4213156868775033450 75/190335427744yz20+963092838906357780\
   203775/14465492508544z21+5783119609876033908806 27685/58623311745152yz21+7\
   9736582933043680846398765 5/29311655872576z22-9613866436536887157049371325\
   /13526174813184yz22+501240934527722519528081723588067 5/40098345233683968z\
   23-5797537140993441092129362614650697 75/35642973541052416z24
↦ j[134]=xy6z+y7z+y6z2+xyz6+y2z6+yz7-355/2y10z-1205/2y9z2-3375/2y8z3-2775y7\
   z4-2415y6z5-315y5z6+1725y4z7+4125/2y3z8+2295/2y2z9+745/2yz10+100z11+24965\
   /72y10z3-27195/16y8z5-85985/9y7z6-876415/36y6z7-221585/6y5z8-5217875/144y\
   4z9-1685885/72y3z10-462455/48y2z11-135245/48yz12-36005/72z13+90156375/16y\
   7z7+29671452975/1504y6z8+51237159525/1504y5z9+593233060725/16544y4z10+858\
   1894275/352y3z11+175264666425/16544y2z12+52418668725/16544yz13+4569203775\
   /8272z14+4527194666704915/377898048y6z9-91063044331423475/2771252352y4z11\
   -1237615165679062475/17666733744y3z12-1576377074570557715/23555644992y2z1\
   3-534894131835638785/15703763328yz14-312627981834621175/35333467488z15+26\
   909968131759537225/19775109376y3z13+1414479550383030080625/3045366843904y\
   2z14+1414490856307438074225/3045366843904yz15-2729651088058929442425/3045\
   366843904z16-3260038437503090421268275/3245581273088y2z15-326003843750309\
   0421268275/3245581273088yz16-3260038437503090421268275/3245581273088z17-3\
   22825/36yz17+2126975/288z18+1100669625/94yz18-11440903875/1504z19-4693998\
   406299868175/12848533632yz19-1767548653539102175/4282844544z20+3751356791\
   6654865942125/4568050265856yz20-14919266617134945384233375/34717182020505\
   6z21-6871474217877063073648628625/937972987922432yz21-7625237430178998007\
   4234957625/3751891951689728z22+661763841336628411346870582875/97388458654\
   9248yz22-3808640826617296706574614083873538125/46193293709203931136z23+46\
   83340666688735257491327168397881875/427715682492628992z24
↦  j[135]=x2y6+2xy7+y8-y6z2-2xyz6-2y2z6-2yz7+355y10z+1205y9z2+3375y8z3+5550y\
   7z4+4830y6z5+630y5z6-3450y4z7-4125y3z8-2295y2z9-745yz10-200z11-7880/9y10z\
   3+4125y8z5+182776/9y7z6+443696/9y6z7+219626/3y5z8+639835/9y4z9+411086/9y3\
   z10+55879/3y2z11+16273/3yz12+8768/9z13-11290212y7z7-1857865224/47y6z8-320\
   6045637/47y5z9-37085969520/517y4z10-535648557/11y3z11-10913086818/517y2z1\
   2-3256579701/517yz13-567589182/517z14-4527226432785547/188949024y6z9+9106\
   3826265715955/1385626176y4z11+1222601537677268075/8833366872y3z12+1546341\
   988228093127/11777822496y2z13+514866309979671673/7851881664yz14+297605578\
   453052185/17666733744z15-26650645451955170685/9887554688y3z13-13546032070\
   76200144365/1522683421952y2z14-1354614513000608137965/1522683421952yz15+2\
   749591738675886931525/1522683421952z16+3260024493763254599915091/16227906\
   36544y2z15+3260024493763254599915091/1622790636544yz16+326002449376325459\
   9915091/1622790636544z17+164030/9yz17-236035/18z18-2162488815/94yz18+1464\
   107115/94z19+4613994316675472615/6424266816yz19+1767875680323979615/21414\
   22272z20-24781578509803143452225/2284025132928yz20+1538592913564430065729\
   1075/173585910102528z21+6850490564209627802665518345/468986493961216yz21+\
   76210039283786401433405282505/1875945975844864z22-65547918385969582582762\
   4430175/486942293274624yz22+3808773256633029793294610128016151325/2309664\
   6854601965568z23-4683314916222080708082298626954441725/213857841246314496\
   z24
↦  j[136]=x2z5+2xyz5+y2z5+2xz6+2yz6+z7-40/3y10z2-200/3y9z3-210y8z4-440y7z5-5\
   60y6z6-420y5z7-140y4z8+40y3z9+60y2z10+70/3yz11+50/3z12-3080/9y7z7-4597120\
   /3807y6z8-2642860/1269y5z9-2765600/1269y4z10-118520/81y3z11-262340/423y2z\
   12-230000/1269yz13-122080/3807z14-43358150/19881y6z9+542829800/72897y4z11\
   +50774168200/3717747y3z12+16314938050/1239249y2z13+2362277150/413083yz14+\
   6278620000/3717747z15-127759277025000/77246521y3z13-1388927940002550/5407\
   25647y2z14-1388927940002550/540725647yz15-44964818257050/49156877z16-9666\
   5296634185225/1728825252y2z15-96664518662821825/1728825252yz16-9666477798\
   6609625/1728825252z17-799400/1269yz18+2186300/3807z19+197714388625/371774\
   7yz19+108882107750/1239249z20-2918681078960250/49156877yz20-1396246436249\
   4000/540725647z21+3444272968024528433621125/113915753504784yz21+291115180\
   3095380873115875/170873630257176z22+766509635493484109048125/29662664064y\
   z22+29705816576394232481138125/5329391976832z23+4500153574996703406538290\
   7460375/73817408271100032z24
↦  j[137]=xy2z4+y3z4+xyz5+2y2z5+yz6-20y10-100y9z-315y8z2-660y7z3-840y6z4-630\
   y5z5-210y4z6+60y3z7-135xyz8-45y2z8-100yz9+25z10-1610/9y10z2-5140/9y9z3-96\
   5y8z4-1480/3y7z5+3710/3y6z6+3080y5z7+10370/3y4z8+6980/3y3z9+925y2z10+2300\
   /9yz11+385/9z12+87975/2y10z3+840375/4y8z5+879840y7z6+1538415y6z7+1474470y\
   5z8+3120975/4y4z9+326205/2y3z10+33345/4y2z11-44685/4yz12-17235/2z13+38563\
   8815/108y7z7+1141900440815/91368y6z8+416748918215/30456y5z9+90281767525/3\
   0456y4z10-22012332455/1944y3z11-139084686605/10152y2z12-199537054955/3045\
   6yz13-54418120115/45684z14-1912706515/59643y6z9+88840501375/874764y4z11+1\
   7549668629800/59643y3z12+35091245489405/79524y2z13+32159571656840/72897yz\
   14+35082820132475/238572z15-331518523667152725/72702608y3z13-992594778772\
   803975/145405216y2z14-992594778772803975/145405216yz15-329557731438498525\
   /145405216z16+113555964610605234325/11854801728y2z15+11355597192106629992\
   5/11854801728yz16+113555948902992944725/11854801728z17-1780425yz17-718447\
   5/8z18-272059126825/3807yz18-5712653134075/91368z19+2827267021747775/1312\
   146yz19-21358112205325/874764z20-70125515834268688875/72702608yz20-135173\
   867341968480375/290810432z21+23183458241171746356470875/2871825718608yz21\
   +578558833987552408530674875/137847634493184z22-7873849209987716152157851\
   25/397828670976yz22-6966530415752642565239071625/40843743553536z23-313607\
   3914867803773218240136443375/29775089050527744z24
↦ j[138]=x2yz4-y3z4-2y2z5-yz6+40y10+200y9z+630y8z2+1320y7z3+1680y6z4+1260y5\
   z5+420y4z6-120y3z7+270xyz8+90y2z8+200yz9-50z10+3280/9y10z2+10490/9y9z3+19\
   90y8z4+3080/3y7z5-7840/3y6z6-6580y5z7-22420/3y4z8-15280/3y3z9-2060y2z10-5\
   110/9yz11-890/9z12-87975y10z3-840375/2y8z5-1759680y7z6-3076830y6z7-294894\
   0y5z8-3120975/2y4z9-326205y3z10-33345/2y2z11+44685/2yz12+17235z13-3855939\
   35/54y7z7-1141769232815/45684y6z8-416674662935/15228y5z9-90204989125/1522\
   8y4z10+22015593095/972y3z11+139091906525/5076y2z12+199543535435/15228yz13\
   +54419763635/22842z14+3719602805/59643y6z9-43563970400/218691y4z11-656349\
   8116151800/11153241y3z12-6562045593979985/7435494y2z13-2186854806328795/2\
   478498yz14-6560551756919225/22306482z15+5636592788190616575/617972168y3z1\
   3+118130666021815191975/8651610352y2z14+118130666021815191975/8651610352y\
   z15+39218366987146559925/8651610352z16-404241320171262378095/20745903024y\
   2z15-404241355093532468495/20745903024yz16-404241268306504818095/20745903\
   024z17+3560850yz17+7184475/4z18+544122978050/3807yz18+5712586856875/45684\
   z19-96126468962468975/22306482yz19+181445144883700/3717747z20+83452069528\
   37589967875/4325805176yz20+16085912822731650765375/17303220704z21-1105684\
   7052455623089489713975/683494521028704yz21-4313891426610580568005726325/5\
   12620890771528z22+13297493452729982339683359625/3381543703296yz22+9147355\
   8207040733909350464875/303775342679424z23+3795190175377956333920584649269\
   08125/1771617798506400768z24
↦ j[139]=xy3z3+y4z3+y3z4-xyz5-y2z5-yz6+30y10+135y9z+405y8z2+720y7z3+630y6z4\
   -630y4z6-720y3z7+270xyz8-135y2z8+150yz9-45z10+390y10z2+1495y9z3+3105y8z4+\
   3570y7z5+1470y6z6-1890y5z7-3630y4z8-2895y3z9-1215y2z10-335yz11-65z12-4893\
   75/8y10z3-7660575/16y8z5-1404405y7z6-6436395/4y6z7-757755/2y5z8+17697825/\
   16y4z9+11534535/8y3z10+11875815/16y2z11+3904605/16yz12+414855/8z13-186570\
   055/48y7z7-184150364285/13536y6z8-67208038085/4512y5z9-14559824975/4512y4\
   z10+3549863045/288y3z11+22429897495/1504y2z12+32179064545/4512yz13+877588\
   2185/6768z14+1970986785/8836y6z9-283481868375/388784y4z11-265595666951625\
   /826166y3z12-3181635711832995/6609328y2z13-794204130017655/1652332yz14-10\
   58349135631575/6609328z15+24762740108314000725/4943777344y3z13+5191852859\
   03995733025/69212882816y2z14+519185285903995733025/69212882816yz15+172506\
   924387599722875/69212882816z16-475892294042319376455/36881605376y2z15-475\
   892344754526768455/36881605376yz16-475892218435028355655/36881605376z17+1\
   5884775/4yz17+26973675/32z18+43874244925/564yz18+921266693425/13536z19-77\
   65087811458575/3304664yz19+146456942381175/6609328z20+3636342552610189369\
   0125/34606441408yz20+70081263027558240944625/138425765632z21-694538650194\
   682283342883775/75943835669856yz21-3826744152882804943302504225/810067580\
     478464z22+5697616275288902757071551125/3005816625152yz22-1493873395012718\
     830990514070375/6480540643827712z23+68320986527833081341219282225285875/5\
     24923792150044672z24
↦ j[140]=x2y2z3-y4z3-2y3z4-y2z5-20y10-70y9z-180y8z2-120y7z3+420y6z4+1260y5z\
     5+1680y4z6+1320y3z7-270xyz8+360y2z8-100yz9+40z10-3920/9y10z2-17140/9y9z3-\
     4490y8z4-19780/3y7z5-17920/3y6z6-2800y5z7+620/3y4z8+3530/3y3z9+640y2z10+1\
     640/9yz11+460/9z12+137475/4y10z3+4299075/8y8z5+1049130y7z6+282735/2y6z7-2\
     191185y5z8-30181725/8y4z9-12839355/4y3z10-12009195/8y2z11-3725865/8yz12-3\
     45915/4z13+136469635/216y7z7+404149783795/182736y6z8+147447073195/60912y5\
     z9+31808151425/60912y4z10-7804613995/3888y3z11-49285406905/20304y2z12-707\
     01736015/60912yz13-19280184295/91368z14-45738581885/119286y6z9+2005208667\
     25/159048y4z11+607622992216775/11153241y3z12+2386776226181815/29741976y2z\
     13+97886617671010/1239249yz14+2333597351728825/89225928z15-22233918786099\
     82425/2471888672y3z13-46803411748827695925/34606441408y2z14-4680341174882\
     7695925/34606441408yz15-15675925448287941975/34606441408z16+9723126342598\
     5774145/15087929472y2z15+97231293098913735745/15087929472yz16+97231246074\
     866881345/15087929472z17-8763075/2yz17+1764225/16z18-96373577075/7614yz18\
     -2023431835775/182736z19+17404765279775075/44612964yz19+135884742919025/2\
     9741976z20-2984753291459929115625/17303220704yz20-5740829192649594151125/\
     69212882816z21+16891425512760557317733275/7766983193508yz21+3186606302672\
     240245483256975/2982521546307072z22+2613640829282507187241228375/13526174\
     813184yz22+702584003191650911266872000875/883710087794688z23-305550300702\
     59092298956649239970375/644224654002327552z24
↦ j[141]=xy4z2+y5z2+y4z3+xyz5+y2z5+yz6-50y10-220y9z-720y8z2-1380y7z3-1470y6\
     z4-630y5z5+420y4z6+780y3z7-405xyz8+90y2z8-250yz9+55z10-9475/18y10z2-35615\
     /18y9z3-7765/2y8z4-11515/3y7z5-385/3y6z6+4865y5z7+20585/3y4z8+30205/6y3z9\
     +4105/2y2z10+9865/18yz11+970/9z12+841275/8y10z3+11022075/16y8z5+2284245y7\
     z6+12590055/4y6z7+3706695/2y5z8-5213925/16y4z9-10229715/8y3z10-11742435/1\
     6y2z11-4083345/16yz12-483795/8z13+3221955035/432y7z7+9540448435115/365472\
     y6z8+3482057820995/121824y5z9+754702446025/121824y4z10-183876096995/7776y\
     3z11-1161902722625/40608y2z12-1666944112775/121824yz13-454611458495/18273\
     6z14-124282787435/477144y6z9+739024544875/874764y4z11+6867618145472450/11\
     153241y3z12+54884202969283745/59483952y2z13+1661634827279765/1802544yz14+\
     54817401101130725/178451856z15-47304297103281151275/4943777344y3z13-99159\
     8478668958395475/69212882816y2z14-99159847866895839475/69212882816yz15-3\
     29338319223022277625/69212882816z16+3771274126334450391515/165967224192y2\
     z15+3771274434065345247515/165967224192yz16+3771273589015562069915/165967\
     224192z17-23006475/4yz17-55711575/32z18-2272827028975/15228yz18-477252113\
     03575/365472z19+803845315225396325/178451856yz19-172615641033050/3717747z\
     20-69741387426522272502375/34606441408yz20-134419242103963735642875/13842\
     5765632z21+94059878420829155387538074825/5467956168229632yz21+73132107182\
     695124138089059175/8201934252344448z22-105252038098981778677218479875/270\
     52349626368yz22+175858686084750512522800802125/2430202741435392z23-177613\
     496876320718116620695895696875/745944336213221376z24
↦ j[142]=xy5z+y6z+y5z2-xyz5-y2z5-yz6+75y10+255y9z+810y8z2+1440y7z3+1260y6z4\
     -1260y4z6-1440y3z7+540xyz8-270y2z8+285yz9-75z10+2365/4y10z2+9385/4y9z3+18\
     615/4y8z4+10215/2y7z5+3675/2y6z6-6015/2y5z7-5310y4z8-16385/4y3z9-6565/4y2\
     z10-1575/4yz11-175/2z12-489375/4y10z3-7660575/8y8z5-2808810y7z6-6436395/2\
     y6z7-757755y5z8+17697825/8y4z9+11534535/4y3z10+11875815/8y2z11+3904605/8y\
     z12+414855/4z13-62232405/8y7z7-184274607085/6768y6z8-67278492205/2256y5z9\
     -14632785575/2256y4z10+3546760885/144y3z11+22423028815/752y2z12+321729180\
     25/2256yz13+8774314105/3384z14+4178734545/8836y6z9-296439626475/194392y4z\
     11-265646129182350/413083y3z12-3181941699916995/3304664y2z13-397112778283\
     365/413083yz14-1058339974474425/3304664z15+247476976379356569 75/247188867\
     2y3z13+518798480042056116825/34606441408y2z14+518798480042056116825/34606\
     441408yz15+172330713110956919175/34606441408z16-999933682115336447375/368\
     81605376y2z15-999933733058053872975/36881605376yz16-999933566168789546575\
     /36881605376z17+15884775/2yz17+26973675/16z18+21935974750/141yz18+9213289\
     88225/6768z19-15532249821596025/3304664yz19+72106910279025/1652332z20+181\
     7766154239521560 7625/8651610352yz20+70062861745703586013875/69212882816z2\
     1-1100901412609598696579506 9375/607550685358848yz21-227861224565460309271\
     97291125/2430202741435392z22+296707534564631017623877 8375/751454156288yz2\
     2-3225761466861633204571190657625/648054064382 7712z23+720568196235118700 9\
     5854329883292375/262461896075022336z24
  ↦  j[143]=x2y5+2xy6+y7-y5z2+2xyz5+2y2z5+2yz6-150y10-510y9z-1620y8z2-2880y7z3\
     -2520y6z4+2520y4z6+2880y3z7-1080xyz8+540y2z8-570yz9+150z10-3115/3y10z2-12\
     245/3y9z3-7635y8z4-7910y7z5-2450y6z6+4890y5z7+7960y4z8+5855y3z9+2165y2z10\
     +1285/3yz11+320/3z12+489375/2y10z3+7660575/4y8z5+5617620y7z6+6436395y6z7+\
     1515510y5z8-17697825/4y4z9-11534535/2y3z10-11875815/4y2z11-3904605/4yz12-\
     414855/2z13+560392165/36y7z7+1659353953285/30456y6z8+606007559485/10152y5\
     z9+132214585175/10152y4z10-31898741005/648y3z11-201758312575/3384y2z12-28\
     9512553945/10152yz13-78957630385/15228z14-38835269005/39762y6z9+455305413\
     325/145794y4z11+4782127765547200/3717747y3z12+9546317921362435/4956996y2z\
     13+288813258706345/150212yz14+9525007244173975/14870988z15-24736385490900\
     216975/1235944336y3z13-7407084055107108352 5/2471888672y2z14-7407084055107\
     1083525/2471888672yz15-24598069569270649575/2471888672z16+223734656037135\
     329315/3951600576y2z15+223734658506885689315/3951600576yz16+2237346345997\
     02204515/3951600576z17-15884775yz17-26973675/8z18-394830950375/1269yz18-8\
     292401068025/30456z19+139801227165196975/14870988yz19-107149749173350/123\
     9249z20-519282472823646140587 5/1235944336yz20-10007069484819215984625/494\
     3777344z21+58591462290317531301087 2275/1627367907 2112yz21+145736335691660\
     82540991174825/781136595461376z22-18298138219439992742845470875/225436246\
     8864yz22+25438802886785768252951386 1125/231447880136704z23-9704984454692 2\
     2129506478201283956 25/1687255046196572 16z24
  ↦  j[144]=x2z4+2xyz4+y2z4+2xz5+2yz5+z6-40/3y10z-200/3y9z2-210y8z3-440y7z4-56\
     0y6z5-420y5z6-140y4z7+40y3z8+60y2z9+70/3yz10+50/3z11-970/27y10z3+275y8z5-\
     880/27y7z6-44660/27y6z7-33440/9y5z8-116875/27y4z9-83330/27y3z10-12355/9y2\
     z11-3745/9yz12-1970/27z13+622575y7z7+204659975/94y6z8+354074625/94y5z9+41\
     11313025/1034y4z10+59810975/22y3z11+1231645125/1034y2z12+371410425/1034yz\
     13+32045225/517z14+804396076007015/283423536y6z9-15602797815376975/207843\
     9264y4z11-204559448538787975/13250050308y3z12-260647970458585315/17666733\
     744y2z13-8534945935 1920685/11777822496yz14-57064091716957925/26500100616z\
     15+1547893757622142875/4943777344y3z13-135502793998762454645/761341710976\
     y2z14-135502451394992515445/761341710976yz15-373878204270059164595/761341\
     710976z16-334770039100568948609925/811395318272y2z15-33477003910056894860\
     9925/811395318272yz16-334770039100568948609925/811395318272z17-45650/27yz\
     17+70375/54z18+65218875/47yz18-73451875/94z19-822175819836802675/96364002\
     24yz19-360821137006901675/3212133408z20-47899191649305383420 75/3426037699\
     392yz20-26195226055089377412392 5/13704150797568z21-607807101752318769477 4\
     65375/234493246980608yz21-669406553281043692580757 3375/937972987922432z22\
     +11429005956704163769705140687 5/730413439911936yz22-12088415616114669001 7\
     9313939139894625/3464497028190294835 2z23+14428800289927937556114127220075\
     61625/3207867618694717 44z24
  ↦  j[145]=x2yz3+2xy2z3+y3z3+2xyz4+2y2z4+yz5+20/3y10z+70/3y9z2+60y8z3+40y7z4-\
     140y6z5-420y5z6-560y4z7-440y3z8-210y2z9-170/3yz10-40/3z11+1640/27y10z3-55\
     0y8z5-880/27y7z6+80080/27y6z7+61600/9y5z8+217250/27y4z9+155620/27y3z10+23\
     150/9y2z11+7130/9yz12+3700/27z13-616860y7z7-101537270/47y6z8-176449860/47\
     y5z9-2060746950/517y4z10-30253460/11y3z11-631897590/517y2z12-193182480/51\
     7yz13-34023460/517z14-390581156629105/283423536y6z9+7957951023326825/2078\
     439264y4z11+119009654874548225/13250050308y3z12+158491780449444005/176667\
     33744y2z13+60566131167443995/11777822496yz14+36237699069890875/2650010061\
     6z15-836750902787587725/4943777344y3z13-91984607852349272765/761341710976\
     y2z14-91984950456119211965/761341710976yz15+36874916975682590485/76134171\
     0976z16+68771109701305346798115/811395318272y2z15+68771109701305346798115\
     /811395318272yz16+68771109701305346798115/811395318272z17+87550/27yz17-66\
     625/27z18-71294475/47yz18+30843475/47z19+457852986500046725/9636400224yz1\
     9+144051438426095725/3212133408z20-13333482334520062104275/3426037699392y\
     z20+580959394913905815587225/260378865153792z21+1665561261868153075001864\
     25/234493246980608yz21+1817794953847119935326608825/937972987922432z22-61\
     781302578779726094039662125/730413439911936yz22+2346631141984570356154105\
     45355446375/34644970281902948352z23-296389892082073496998676969809364375/\
     320786761869471744z24
   ↦ j[146]=xy3z2+y4z2+3/2xy2z3+5/2y3z3+1/2xyz4+2y2z4+1/2yz5+135/2xy8+105/2y9-\
     270y7z2-630y6z3-945y5z4-945y4z5-630y3z6+135xyz7-135y2z7+135/2xz8+135yz8+6\
     0z9-40/3y10z-110/3y9z2-285/2y8z3-320y7z4-455y6z5-420y5z6-245y4z7-80y3z8-1\
     5/2y2z9+10/3yz10-5/6z11+72900y9z3+328050y8z4+1482300y7z5+3657150y6z6+5503\
     950y5z7+5382450y4z8+3474900y3z9+1439775y2z10+407025yz11+66825z12-250/27y1\
     0z3+275/3y8z5-880/27y7z6-16940/27y6z7-12320/9y5z8-42625/27y4z9-30230/27y3\
     z10-4465/9y2z11-1375/9yz12-710/27z13-8869950y7z7-4339409375/141y6z8-24669\
     75200/47y5z9-28183325750/517y4z10-1202702000/33y3z11-8022493800/517y2z12-\
     2390131100/517yz13-1217165875/1551z14+15262246589151325/566847072y6z9-290\
     889065829543125/4156878528y4z11-3265891297020894125/26500100616y3z12-2910\
     726002122409825/35333467488y2z13-292112628380862175/23555644992yz14+73867\
     3534772524625/53000201232z15-40405500316963449975/9887554688y3z13-7374928\
     14593884236065/217526203136y2z14-737492765650488530465/217526203136yz15+1\
     51428230446397212185/217526203136z16+404585865449881395260175/23182723379\
     2y2z15+404586140078218226108175/231827233792yz16+404585590821544564412175\
     /231827233792z17-16400/27yz17+25825/54z18-1362147875/94yz18+4167859625/28\
     2z19-10307586092286934625/19272800448yz19-2542176954710379625/6424266816z\
     20-350710109981330125635775/978867914112yz20-6761375244076533980868275/74\
     393961472512z21+182682706626437703146166858375/1272963340751872yz21+47657\
     4285477196273600205170375/5091853363007488z22+176705782072543357174463400\
     981625/1460826879823872yz22+2090200156634672248126068543685731875/9898562\
     937686556672z23-5230960393693986371599262407509219625/274960081602404352z\
     24
   ↦ j[147]=x2y2z2-y4z2-xy2z3-3y3z3-xyz4-3y2z4-yz5-135xy8-105y9+540y7z2+1260y6\
     z3+1890y5z4+1890y4z5+1260y3z6-270xyz7+270y2z7-135xz8-270yz8-120z9+40/3y10\
     z+50/3y9z2+75y8z3+200y7z4+350y6z5+420y5z6+350y4z7+200y3z8+75y2z9+50/3yz10\
     +25/3z11-145800y9z3-656100y8z4-2964600y7z5-7314300y6z6-11007900y5z7-10764\
     900y4z8-6949800y3z9-2879550y2z10-814050yz11-133650z12-290/27y10z3+275/3y8\
     z5+880/27y7z6-10780/27y6z7-8800/9y5z8-31625/27y4z9-22870/27y3z10-3425/9y2\
     z11-1055/9yz12-550/27z13+18362475y7z7+17971617425/282y6z8+10221975425/94y\
     5z9+116844616025/1034y4z10+4990240925/66y3z11+33321620325/1034y2z12+99319\
     34825/1034yz13+2530467425/1551z14-7228924937190235/141711768y6z9+13764312\
     6145374275/1039219632y4z11+1530665828395767275/6625025154y3z12+1325038902\
     868983935/8833366872y2z13+103381553755535065/5888911248yz14-3978688208607\
     71675/13250050308z15+20976698235384460425/2471888672y3z13+359067654365364\
     209535/54381550784y2z14+359067654365364209535/54381550784yz15-10241970953\
     2171459015/54381550784z16-226205088423035811404625/57956808448y2z15-22620\
     5225737204226828625/57956808448yz16-2262049511088673959806 25/57956808448z\
     17-13300/27yz17+19625/54z18+1427366750/47yz18-8556074875/282z19+474270489\
     9011403575/4818200112yz19+1090677832892444575/1606066704z20+1750129180885\
     89866221225/244716978528yz20+30251807638129208332657 25/18598490368128z21-\
     921662363150320256106360956 25/318240835187968yz21-2473719558725776312 2953\
     1865625/1272963340751872z22-8829574741088917387347801030 7375/365206719955\
     968yz22-1131445939795162936725289916890416125/247464073442163916 8z23+2924\
     668919731023673319941364314212875/68740020400601088z24
  ↦ j[148]=xy4z+y5z+y4z2-5/2xy2z3-5/2y3z3-3/2xyz4-4y2z4-3/2yz5-135/2xy8-75/2y\
     9+270y7z2+630y6z3+945y5z4+945y4z5+630y3z6-135xyz7+135y2z7-135/2xz8-135yz8\
     -75z9+335/6y10z+745/6y9z2+330y8z3+545y7z4+560y6z5+315y5z6+20y4z7-215/2y3z\
     8-80y2z9-275/6yz10+5/6z11-72900y9z3-328050y8z4-1482300y7z5-3657150y6z6-55\
     03950y5z7-5382450y4z8-3474900y3z9-1439775y2z10-407025yz11-66825z12-7415/1\
     08y10z3+11825/24y8z5+34540/27y7z6+92785/54y6z7+12320/9y5z8+130625/216y4z9\
     +6935/108y3z10-6355/72y2z11-3025/72yz12-985/108z13+67915005/8y7z7+6641580\
     6335/2256y6z8+37734996335/752y5z9+430823869175/8272y4z10+18372277355/528y\
     3z11+122452925715/8272y2z12+36489901055/8272yz13+9271375415/12408z14-2824\
     9026130984625/1133694144y6z9+5397441514538376 25/8313757056y4z11+604854025\
     8580293025/53000201232y3z12+523402906879548092 5/70666934976y2z13+43080252\
     0958574075/47111289984yz14-16047529707203022 25/106000402464z15+8479470305\
     5556652075/19775109376y3z13+909035625576590191 9435/3045366843904y2z14+909\
     03551518204210042 35/3045366843904yz15-39680281289910791465 15/304536684390\
     4z16-71272177554352954396838 85/3245581273088y2z15-71272216002320110715558\
     85/3245581273088yz16-712721391063857980781 1885/3245581273088z17-39875/54y\
     z17+19075/432z18+2580029275/188yz18-32172174175/2256z19+18543205116623621\
     125/38545600896yz19+3892516801482312125/12848533632z20+482116417509208855\
     6344725/13704150797568yz20+70879674286195701926458225/1041515460615168z21\
     -260376905004788204951174443792 5/17821486770526208yz21-7181820093185285 28\
     672502922232 5/712859470821048 32z22-35311675265874889420819367283 3125/2921\
     653759647744yz22-34620701793671529463257256951523943625/13857988112761179\
     3408z23+921512739642072460780295944089815693 75/3849441142433660928z24
  ↦ j[149]=x2y3z-y5z-2y4z2+2xy2z3+y3z3+2xyz4+4y2z4+2yz5-30y9+30z9-205/3y10z-4\
     55/3y9z2-315y8z3-410y7z4-350y6z5-210y5z6-110y4z7-65y3z8-35y2z9+55/3yz10-4\
     0/3z11+9715/54y10z3-5775/4y8z5-52360/27y7z6+35035/27y6z7+61600/9y5z8+1029\
     875/108y4z9+398405/54y3z10+124775/36y2z11+39245/36yz12+10325/54z13+565815\
     /4y7z7+188885535/376y6z8+320921805/376y5z9+3613595925/4136y4z10+49080555/\
     88y3z11+896742345/4136y2z12+228233565/4136yz13+21177615/2068z14-305666449\
     2587435/566847072y6z9+5795074703987027 5/4156878528y4z11+715089211641540 47\
     5/26500100616y3z12+89205296916936353 5/35333467488y2z13+266314412886977465\
     /23555644992yz14+193699160527889525/53000201232z15-562172575944033157 5/98\
     87554688y3z13+10587644324485287815 45/1522683421952y2z14+105876416597892 99\
     39945/1522683421952yz15+192450955226188551449 5/1522683421952z16+160055710\
     3319518041837345/1622790636544y2z15+160055710331951804183734 5/16227906365\
     44yz16+1600557103319518041837345/1622790636544z17+143125/27yz17-705575/21\
     6z18+7007325/94yz18-124832175/376z19+2954762704343741575/19272800448yz19+\
     148007982306795057 5/6424266816z20+673492460993573160875 75/6852075398784yz\
     20+2513352332968827753142007 5/520757730307584z21+27624283258321598489698 5\
     4275/468986493961216yz21+3046035334512626794291663147 5/1875945975844864z2\
     2-4157271331040932785273191303 75/1460826879823872yz22+582728766449807636 2\
     507571219335134125/6928994056380589670 4z23-76652386231539915884212428399 4\
     978625/712859470821048 32z24
  ↦ j[150]=x2y4+2xy5+y6-y4z2+5xy2z3+5y3z3+3xyz4+8y2z4+3yz5+135xy8+75y9-540y7z\
     2-1260y6z3-1890y5z4-1890y4z5-1260y3z6+270xyz7-270y2z7+135xz8+270yz8+150z9\
     -180y10z-480y9z2-1245y8z3-1980y7z4-1890y6z5-840y5z6+270y4z7+630y3z8+385y2\
     z9+170yz10+15z11+145800y9z3+656100y8z4+2964600y7z5+7314300y6z6+11007900y5\
     z7+10764900y4z8+6949800y3z9+2879550y2z10+814050yz11+133650z12+2645/9y10z3\
     -12925/6y8z5-51040/9y7z6-70070/9y6z7-19360/3y5z8-56375/18y4z9-5465/9y3z10\
     +495/2y2z11+895/6yz12+295/9z13-31451925/2y7z7-30731921575/564y6z8-1744354\
     5775/188y5z9-198949910575/2068y4z10-8474089675/132y3z11-56405786475/2068y\
     2z12-16813588975/2068yz13-4260508975/3102z14+2333011233040465/47237256y6z\
     9-44713320147338225/346406544y4z11-5038364439578722225/2208341718y3z12-436\
     850618452366265/2944455624y2z13-37858264799994235/1962970416yz14+13426040\
     9947911575/4416683436z15-21262930809107209125/2471888672y3z13-20932125309\
     83993114285/380670855488y2z14-2093211998044795431085/380670855488yz15+118\
     1278870719145414165/380670855488z16+1913673142727368903286775/40569765913\
     6y2z15+1913674103926547811254775/405697659136yz16+19136721815281899953187\
     75/405697659136z17+28000/9yz17-2725/36z18-1172786000/47yz18+15092864375/5\
     64z19-1533811558030297925/1606066704yz19-303281684848221925/535355568z20-\
     397657221580578852065825/571006283232yz20-5156907945035055429009325/43396\
     477525632z21+654656068350189256022553636375/2227685846315776yz21+18362067\
     04848078782812716137375/8910743385263104z22+29424766094977536283310388897\
     625/121735573318656yz22+3048882893746933007429737359173407625/57741617136\
     50491392z23-24743068026622570435918709300895372875/481180142804207616z24
  ↦ j[151]=x2z3+2xyz3+y2z3+2xz4+2yz4+z5-40/3y10-200/3y9z-210y8z2-440y7z3-560y\
     6z4-420y5z5-140y4z6+40y3z7-90xyz8-30y2z8-200/3yz9+50/3z10-3400/27y10z2-11\
     000/27y9z3-2200/3y8z4-4400/9y7z5+6160/9y6z6+6160/3y5z7+22000/9y4z8+15400/\
     9y3z9+2120/3y2z10+5320/27yz11+1040/27z12+29325y10z3+280125/2y8z5+586560y7\
     z6+1025610y6z7+982980y5z8+1040325/2y4z9+108735y3z10+11115/2y2z11-14895/2y\
     z12-5745z13+385575455/162y7z7+1141715096975/137052y6z8+416643978215/45684\
     y5z9+90173223925/45684y4z10-22016943335/2916y3z11-139094896205/15228y2z12\
     -199546212635/45684yz13-54420445715/68526z14-3820549205/178929y6z9+449196\
     93875/656073y4z11+6563624218574800/33459723y3z12+6562127258358785/2230648\
     2y2z13+2186866662922345/7435494yz14+6560582299163225/66919446z15-17083586\
     7641983275/56179288y3z13-39384525701056475025/8651610352y2z14-39384525701\
     056475025/8651610352yz15-13075802084191050675/8651610352z16+4010280179343\
     44591165/62237709072y2z15+401028062192271042365/62237709072yz16+401027972\
     293357938365/62237709072z17-1186950yz17-2394825/4z18-544124966750/11421yz\
     18-5712559667275/137052z19+8738850163016725/6083586yz19-16472642751575/10\
     13931z20-2781822952867571330625/4325805176yz20-5362148196734027354625/173\
     03220704z21+11077116167014107916983025325/2050483563086112yz21+1728722176\
     3845737483664157475/6151450689258336z22-1200911446248001266815509625/9222\
     39191808yz22-184703865630494748541474634125/1822652056076544z23-376527356\
     559012549437942992113834125/5314853395519202304z24
  ↦ j[152]=x2yz2+2xy2z2+y3z2+2xyz3+2y2z3+yz4+20/3y10+70/3y9z+60y8z2+40y7z3-14\
     0y6z4-420y5z5-560y4z6-440y3z7+90xyz8-120y2z8+100/3yz9-40/3z10+4070/27y10z\
     2+17350/27y9z3+4550/3y8z4+19900/9y7z5+17500/9y6z6+2380/3y5z7-2300/9y4z8-4\
     850/9y3z9-850/3y2z10-2150/27yz11-580/27z12-45825/4y10z3-1433025/8y8z5-349\
     710y7z6-94245/2y6z7+730395y5z8+10060575/8y4z9+4279785/4y3z10+4003065/8y2z\
     11+1241955/8yz12+115305/4z13-136258435/648y7z7-403533206515/548208y6z8-14\
     7098508715/182736y5z9-31448045825/182736y4z10+7819897195/11664y3z11+49319\
     248825/60912y2z12+70732162255/182736yz13+19287875815/274104z14+4549783338\
     5/357858y6z9-2197788227975/5248584y4z11-607534990468475/33459723y3z12-238\
     6686764271415/89225928y2z13-97888515183160/3717747yz14-2333852900106025/2\
     67677784z15+742150475229701475/2471888672y3z13+15617519118501164775/34606\
     441408y2z14+15617519118501164775/34606441408yz15+43201755911449125/286003\
     648z16-1115694128256253963295/497901672576y2z15-1115694529343712427295/49\
     7901672576yz16-111569396228902977 1295/497901672576z17+2921025/2yz17-58807\
     5/16z18+96384519875/22842yz18+2023119134975/548208z19-1740333191367657 5/1\
     33838892yz19-136352211268025/89225928z20+904806437911463126 25/1573020064y\
     z20+191397319292860840 8375/69212882816z21-150777312463871524974555 1075/20\
     50483563086112yz21-35721616659904597959235261225/98423211028133376z22-296\
     598414923733621054196 3375/40578524439552yz22-807618201001547256136480 6202\
     125/29162432897224704z23+357586739360521703748821414156558125/21259413582\
     076809216z24
  ↦  j[153]=x2y2z+2xy3z+y4z+2xy2z2+2y3z2+y2z3-40/3y10-170/3y9z-210y8z2-440y7z3\
     -560y6z4-420y5z5-140y4z6+40y3z7-90xyz8-30y2z8-200/3yz9+20/3z10-3220/27y10\
     z2-11000/27y9z3-2200/3y8z4-4400/9y7z5+6160/9y6z6+6160/3y5z7+22000/9y4z8+1\
     5400/9y3z9+2120/3y2z10+5140/27yz11+1040/27z12+29325y10z3+280125/2y8z5+586\
     560y7z6+1025610y6z7+982980y5z8+1040325/2y4z9+108735y3z10+11115/2y2z11-148\
     95/2yz12-5745z13+385567535/162y7z7+1141691954735/137052y6z8+416630886455/\
     45684y5z9+90159691525/45684y4z10-22017517895/2916y3z11-139096168445/15228\
     y2z12-199547355275/45684yz13-54420735155/68526z14-7799308285/357858y6z9+1\
     83193563275/2624292y4z11+596701280401850/3041793y3z12+298280587035530/101\
     3931y2z13+4373743846329215/14870988yz14+596415374726125/6083586z15-939578\
     924022409575/308986084y3z13-9846305093149005825/2162902588y2z14-984630509\
     3149005825/2162902588yz15-817313156248034700/540725647z16+157725016793972\
     5541105/248950836288y2z15+1577250340822250741105/248950836288yz16+1577249\
     989524959534705/248950836288z17-1186950yz17-2394825/4z18-544125795650/114\
     21yz18-5712547960075/137052z19+192255957575674075/133838892yz19-724774243\
     443175/44612964z20-347725574620365778875/540725647yz20-167568542442509632\
     125/540725647z21+221432808411913276745833998 25/4100967126172224yz21+13817\
     455626419146953265628852 5/49211605514066688z22-661477079569623273312648 76\
     25/5072315554944yz22-163004235563126001822636202 7375/14581216448612352z23\
     -194881714572306562244658414644041 25/27972912607995801 6z24
  ↦  j[154]=x2y3+2xy4+y5+2xy3z+2y4z+y3z2+50/3y10+70/3y9z+60y8z2+40y7z3-140y6z4\
     -420y5z5-560y4z6-440y3z7+90xyz8-120y2z8+70/3yz9-40/3z10+3080/27y10z2+1420\
     0/27y9z3+3800/3y8z4+17200/9y7z5+16240/9y6z6+2800/3y5z7+400/9y4z8-2600/9y3\
     z9-520/3y2z10-1160/27yz11-400/27z12-45825/4y10z3-1433025/8y8z5-349710y7z6\
     -94245/2y6z7+730395y5z8+10060575/8y4z9+4279785/4y3z10+4003065/8y2z11+1241\
     955/8yz12+115305/4z13-136543555/648y7z7-404363856115/548208y6z8-147567341\
     035/182736y5z9-31931799425/182736y4z10+7799385835/11664y3z11+49273828345/\
     60912y2z12+70691225935/182736yz13+19277576935/274104z14+93621763045/71571\
     6y6z9-1124181957925/2624292y4z11-608133012764450/33459723y3z12-2387870417\
     969365/89225928y2z13-783216077133305/29741976yz14-233360184283892 5/267677\
     784z15+7401606923920637 25/2471888672y3z13+2226087240906967875/4943777344y\
     2z14+2226087240906967875/4943777344yz15+7457658561228404 25/4943777344z16-\
     7518999518617096 0765/35564405184y2z15-751900173151341863 65/35564405184yz1\
     6-7518998807328992 3965/35564405184z17+2921025/2yz17-588075/16z18+96370091\
     075/22842yz18+2023542883775/548208z19-3482841692262827 5/267677784yz19-711\
     82164476575/44612964z20+142035481156802119875/2471888672yz20+248372839402\
     64045625/898868608z21-833076186818059651888154575/1171704893192064yz21-15\
     348100178950461048135 2075/439389334947024z22-211036828156467837355557 2125\
     /40578524439552yz22-332442630772136331964065016 25/130189432576896z23+4689\
     235638663739362660033684517 6375/3037059083153829888z24
  ↦  j[155]=x2z2+2xyz2+y2z2+2xz3+2yz3+z4-2/3xy2z3-2/3y3z3-2/3xyz4-4/3y2z4-2/3y\
     z5-20/3y10z-80/3y9z2-90y8z3-160y7z4-140y6z5+140y4z7+160y3z8+90y2z9+80/3yz\
     10+10z11-830/27y10z3+275y8z5+880/27y7z6-38500/27y6z7-29920/9y5z8-105875/2\
     7y4z9-75970/27y3z10-11315/9y2z11-3485/9yz12-1810/27z13+413325y7z7+1359707\
     25/94y6z8+235722975/94y5z9+2744424075/1034y4z10+40093425/22y3z11+83109997\
     5/1034y2z12+252250575/1034yz13+21991875/517z14+49791120211145/35427942y6z\
     9-981707768208925/259804908y4z11-26915262096774350/3312512577y3z12-174152\
     27294983795/2208341718y2z13-6047140843471955/1472227812yz14-3863048585443\
     400/3312512577z15+99220216387124175/617972168y3z13-184552402694853624 5/95\
     167713872y2z14-1845495476634374645/95167713872yz15-1712542307540857839 5/9\
     5167713872z16-16814191835989165659975/101424414784y2z15-1681419183598916 5\
     659975/101424414784yz16-16814191835989165659975/101424414784z17-43000/27y\
     z17+65075/54z18+45335250/47yz18-45342375/94z19-53204132191387525/12045500\
     28yz19-21036974719117025/401516676z20+3351963680845715244 25/428254712424y\
     z20-2323471088221975068625 75/32547358144224z21-32253993343737504882347625\
     /29311655872576yz21-35463752537214435032794 6125/117246623490304z22+732545\
     6958222560929731798875/91301679988992yz22-60146226775657390873374125 88900\
     9875/43306212852378685 44z23+12078241497674805171302189500227875/668305753\
     8947328z24
  ↦  j[156]=x2yz+2xy2z+y3z+2xyz2+2y2z2+yz3+1/3xy2z3+1/3y3z3+1/3xyz4+2/3y2z4+1/\
     3yz5+45xy8+35y9-180y7z2-420y6z3-630y5z4-630y4z5-420y3z6+90xyz7-90y2z7+45x\
     z8+90yz8+40z9+10/9y10z+20/9y9z2-5y8z3-160/3y7z4-490/3y6z5-280y5z6-910/3y4\
     z7-640/3y3z8-95y2z9-220/9yz10-65/9z11+48600y9z3+218700y8z4+988200y7z5+243\
     8100y6z6+3669300y5z7+3588300y4z8+2316600y3z9+959850y2z10+271350yz11+44550\
     z12+1870/81y10z3-1925/9y8z5+880/81y7z6+100100/81y6z7+75680/27y5z8+265375/\
     81y4z9+189530/81y3z10+28135/27y2z11+8665/27yz12+4490/81z13-6326265y7z7-18\
     580308065/846y6z8-10574680565/282y5z9-120965644625/3102y4z10-5171874545/1\
     98y3z11-11529185295/1034y2z12-10319323445/3102yz13-2632816985/4653z14+140\
     67277731418885/850270608y6z9-267328523142314525/6235317792y4z11-294173585\
     1974370725/39750150924y3z12-2490411503534248985/53000201232y2z13-14541270\
     4549717015/35333467488yz14+832563980759702225/79500301848z15-142645030789\
     52862325/4943777344y3z13-5119706412424471629935/2284025132928y2z14-511970\
     6755028241569135/2284025132928yz15+4901646446391932222 05/761341710976z16+\
     107854767767823455675 5035/811395318272y2z15+107854831847768716206 7035/811\
     395318272yz16+107854703687878195144 3035/811395318272z17+105500/81yz17-162\
     175/162z18-1499168825/141yz18+8738470825/846z19-9024426703351886825/28909\
     200672yz19-2037318893266015825/9636400224z20-2464013188446215782366225/10\
     278113098176yz20-4178984650013097399888 9725/781136595461376z21+4311656607\
     553958248444330786 75/4455371692631552yz21+34977562627682924930183 23747225\
     /534644603115786 24z22+176529455932853120329574124096625/2191240319735808y\
     z22+16074902003602993181695479936506184125/103934910845708845056z23-41834\
     546022107683399753037302991167375/2887080856825245696z24
  ↦  j[157]=x2y2+2xy3+y4+2xy2z+2y3z+y2z2-2/3xy2z3-2/3y3z3-2/3xyz4-4/3y2z4-2/3y\
     z5+10y9-10z9-80/3y9z2-90y8z3-160y7z4-140y6z5+140y4z7+160y3z8+90y2z9+20yz1\
     0+10z11-665/27y10z3+1375/6y8z5-1760/27y7z6-40810/27y6z7-29920/9y5z8-20762\
     5/54y4z9-73735/27y3z10-21805/18y2z11-6695/18yz12-1735/27z13+736795/2y7z7+\
     726842665/564y6z8+419688265/188y5z9+4881546825/2068y4z10+213665245/132y3z\
     11+1473666285/2068y2z12+446844345/2068yz13+116587885/3102z14+467289960822\
     385/283423536y6z9-9127602561460025/2078439264y4z11-120995451431889425/132\
     50050308y3z12-1545020682143338 85/17666733744y2z13-512782976279491 15/11777\
     822496yz14-33453810794614975/26500100616z15+906953634241417525/4943777344\
     y3z13-539120425971308098 35/761341710976y2z14-161735480650938099905/228402\
     5132928yz15-580748326140071838055/2284025132928z16-178847134773299136689 7\
     15/811395318272y2z15-178847134773299136689715/811395318272yz16-1788471347\
     73299136689715/811395318272z17-40150/27yz17+125725/108z18+39832975/47yz18\
     -248449325/564z19-48143895788576832 5/9636400224yz19-20444173641595532 5/32\
     12133408z20-111997908325919023272 5/3426037699392yz20-262010423431472407 68\
     46225/260378865153792z21-331129424884733448349858425/234493246980608yz21-\
     3648918601091100687569986825/937972987922432z22+6697590805725544159757041\
     7125/730413439911936yz22-64354344914474875461673049733370375/34644970281\
     902948352z23+231252870766209060173841738285177O625/962360285608415232z24
   ↦ j[158]=x2z+2xyz+y2z+2xz2+2yz2+z3-2/3xy2z2-2/3y3z2-2/3xyz3-4/3y2z3-2/3yz4-\
     20/3y10-80/3y9z-90y8z2-160y7z3-140y6z4+140y4z6+160y3z7-60xyz8+30y2z8-100/\
     3yz9+10z10-2450/27y10z2-9250/27y9z3-2150/3y8z4-7300/9y7z5-2380/9y6z6+1820\
     /3y5z7+9500/9y4z8+7550/9y3z9+1090/3y2z10+2690/27yz11+580/27z12+54375/4y10\
     z3+851175/8y8z5+312090y7z6+715155/2y6z7+84195y5z8-1966425/8y4z9-1281615/4\
     y3z10-1319535/8y2z11-433845/8yz12-46095/4z13+559449685/648y7z7+1656592613\
     605/548208y6z8+604442226925/182736y5z9+130593991175/182736y4z10-319676320\
     45/11664y3z11-201910849615/60912y2z12-289649124265/182736yz13-78992436625\
     /274104z14-17472459515/357858y6z9+842146700825/5248584y4z11+2390269317429\
     425/33459723y3z12+9544837831361185/89225928y2z13+397105055726965/3717747y\
     z14+9525468506032975/267677784z15-2752812051085923525/2471888672y3z13-824\
     4367425143148375/4943777344y2z14-8244367425143148375/4943777344yz15-27387\
     43322971301325/4943777344z16+2118210649664415728l5/71128810368y2z15+21182\
     1100926006814415/71128810368yz16+2118210377003975984l5/71128810368z17-176\
     4975/2yz17-2997075/16z18-394881744125/22842yz18-8291014780025/548208z19+6\
     9884274190681925/133838892yz19-438520928661025/89225928z20-57726703117712\
     8798875/2471888672yz20-1112473953175012074375/9887554688z21+5993695670872\
     60601425167775/292926223298016yz21+14885829279357080546349916825/14060458\
     718304768z22-16623359014950125055050528375/40578524439552yz22+27477279038\
     4510449804245496125/4166061842460672z23-91503195605793800152461410980742l\
     25/3037059083153829888z24
   ↦ j[159]=x2y+2xy2+y3+2xyz+2y2z+yz2-2/3xy3z-2/3y4z-2/3xy2z2-4/3y3z2-2/3y2z3+\
     10y10+80/3y9z+90y8z2+160y7z3+140y6z4-140y4z6-160y3z7+60xyz8-30y2z8+100/3y\
     z9-20/3z10+2410/27y10z2+9650/27y9z3+2350/3y8z4+8900/9y7z5+5180/9y6z6-700/\
     3y5z7-6700/9y4z8-5950/9y3z9-890/3y2z10-2170/27yz11-500/27z12-54375/4y10z3\
     -851175/8y8z5-312090y7z6-715155/2y6z7-84195y5z8+1966425/8y4z9+1281615/4y3\
     z10+1319535/8y2z11+433845/8yz12+46095/4z13-559569365/648y7z7-165694140336\
     5/548208y6z8-604639142765/182736y5z9-130797216775/182736y4z10+31959013805\
     /11664y3z11+201891765455/60912y2z12+289631931305/182736yz13+78988107665/2\
     74104z14+18058968265/357858y6z9-867303535525/5248584y4z11-239056339916312\
     5/33459723y3z12-9545414450904185/89225928y2z13-794222524233355/7435494yz1\
     4-9525284419337375/267677784z15+2753042527186728525/2471888672y3z13+57728\
     361763254800025/34606441408y2z14+57728361763254800025/34606441408yz15+191\
     857663826406006075/34606441408z16-13640488546587123539 65/497901672576y2z15\
     -13640491008434282387 65/497901672576yz16-1364048696989849371565/497901672\
     576z17+1764975/2yz17+2997075/16z18+394875663325/22842yz18+8291192536825/5\
     48208z19-6988858546888517 5/133838892yz19+436943255084525/89225928z20+4040\
     801094796307983125/17303220704yz20+7787846744821476359625/69212882816z21-\
     1047672181803023795345498225/512620890771528yz21-103846347836692127525249\
     141075/984232110281333 76z22+1667809579593822538126276 3375/40578524439552y\
     z22-12873219998103952013116696 25375/29162432897224704z23+585269477499477 7\
     83471587033435652875/2125941358207680921 6z24
   ↦ j[160]=x2+2xy+y2+2xz+2yz+z2-2/3xy2z-2/3y3z-2/3xyz2-4/3y2z2-2/3yz3-1/3xy2z\
     3-1/3y3z3-1/3xyz4-2/3y2z4-1/3yz5-15xy8-25/3y9+60y7z2+140y6z3+210y5z4+210y\
     4z5+140y3z6-30xyz7+30y2z7-15xz8-30yz8-40/3z9-10/9y10z-20/9y9z2+5y8z3+160/\
     3y7z4+490/3y6z5+280y5z6+910/3y4z7+640/3y3z8+95y2z9+220/9yz10+65/9z11-1620\
     0y9z3-72900y8z4-329400y7z5-812700y6z6-1223100y5z7-1196100y4z8-772200y3z9-\
     319950y2z10-90450yz11-14850z12-1870/81y10z3+1925/9y8z5-880/81y7z6-100100/\
     81y6z7-75680/27y5z8-265375/81y4z9-189530/81y3z10-28135/27y2z11-8665/27yz1\
     2-4490/81z13+2245305y7z7+6598212305/846y6z8+3759783605/282y5z9+4307153082\
     5/3102y4z10+1845502865/198y3z11+4126709415/1034y2z12+3701280965/3102yz13+\
     947136845/4653z14-4520669493739885/850270608y6z9+85503196850075525/623531\
     7792y4z11+919902607150906925/39750150924y3z12+746481393750364385/53000201\
     232y2z13+13136147016481615/35333467488yz14-297400526011588025/79500301848\
     z15+4888765901631375325/4943777344y3z13+1818903882360545256935/2284025132\
     928y2z14+1818904224964315196135/2284025132928yz15-146568629352993015205/7\
     61341710976z16-355094819337109366083795/811395318272y2z15-355095032936926\
     901187795/811395318272yz16-355094605737291830979795/811395318272z17-10550\
     0/81yz17+162175/162z18+548903525/141yz18-3026775025/846z19+27820127662598\
     75825/28909200672yz19+631251774268012825/9636400224z20+834549760512521153\
     551225/10278113098176yz20+14530392406701081768114725/781136595461376z21-1\
     43805121095377365823474118475/4455371692631552yz21-1168361866569917419774\
     687429825/53464460311578624z22-58813484594869708628716919721625/219124031\
     9735808yz22-5298255598048491212044632289975509125/103934910845708845056z2\
     3+13773306630404801119508893030689754375/2887080856825245696z24
```

See also: Section D.4.31.1 [borderBasis], page 1266.

D.4.31.3 rJanet

Procedure from library `rstandard.lib` (see Section D.4.31 [rstandard_lib], page 1265).

Usage: rJanet(I); I is an ideal.

Return: ideal, a Janet basis for I.

Purpose: Computes a Janet basis for the ideal given by the generators in I for any ordering.

Example:

```
  LIB "rstandard.lib";
  ring R= 32003,(a,b,c,d,e,f,g),dp;
  ideal i=
  a+b+c+d+e+f+g,
  ab+bc+cd+de+ef+fg+ga,
  abc+bcd+cde+fde+efg+fga+gab,
  abcd+bcde+cdef+defg+efga+fgab+gabc,
  abcde+bcdef+cdefg+defga+efgab+fgabc+gabcd,
  abcdef+bcdefg+cdefga+defgab+efgabc+fgabcd+gabcde,
  abcdefg-1;
  ideal j = rJanet(i);   j;
  ↦ Length of Janet basis: 210
  ↦ j[1]=a+b+c+d+e+f+g
  ↦ j[2]=b2+bd-cd+be-de+bf-ef+2bg+cg+dg+eg+g2
  ↦ j[3]=bc2-bcd+c2d+cef-def-c2g-ceg+deg+bfg+cfg+dfg+efg+f2g-bg2-2cg2-dg2-eg2\
    +fg2-g3
  ↦ j[4]=c2d2+cd3+bd2e+cd2e+d3e-2bde2-2cde2-de3+2bcdf+cd2f+2cdef-2be2f-ce2f-3\
    de2f-e3f+bef2-cef2+def2-e2f2-2bcdg+2cd2g+2bceg+cdeg+2d2eg+de2g-3bcfg+2bdf\
    g-cdfg+2d2fg-befg-cefg-3defg-e2fg-2cf2g+df2g-4ef2g-f3g+bcg2-c2g2+cdg2-2be\
    g2-3ceg2-2e2g2+5bfg2+5cfg2+7dfg2+4efg2+f2g2-2bg3-3cg3-2dg3-4eg3+5fg3-2g4
  ↦ j[5]=bcd2-bcde+cd2e-cdef+d2ef-cd2g+cdeg-d2eg+bcfg-bdfg-d2fg+befg+cefg+def\
    g+e2fg-df2g+2ef2g-bcg2+bdg2+d2g2-2bfg2-2cfg2-3dfg2-efg2-f2g2+bg3+cg3+2dg3\
    +eg3-2fg3+g4
  ↦ j[6]=c3d+cd3+bcde+2c2de+bd2e-cd2e+d3e-2bde2-2cde2-de3+2bcdf+c2df+cd2f+bce\
    f+c2ef-bdef+2cdef-d2ef-2be2f-4de2f-e3f+bef2-e2f2-c3g-3bcdg+3c2dg+4cd2g+bc\
    eg-2c2eg+bdeg+cdeg+3d2eg-ce2g+2de2g-3bcfg-c2fg+3bdfg+3d2fg-befg+cefg-4def\
    g-e2fg+bf2g-cf2g+3df2g-3ef2g-5c2g2-bdg2+cdg2-d2g2-3beg2-8ceg2+deg2-3e2g2+\
    9bfg2+7cfg2+11dfg2+6efg2+6f2g2-5bg3-9cg3-6dg3-8eg3+8fg3-5g4
  ↦ j[7]=bde2f-11823cde2f-7526d2e2f-2154be3f+7864ce3f+15572de3f+15266e4f+1331\
    0c3f2+12479bcdf2+8143c2df2+4270bd2f2+10092cd2f2-5691d3f2-14681bcef2+14474\
    c2ef2-11522bdef2-179cdef2+14676d2ef2-2804be2f2+5761ce2f2+5413de2f2-9491e3\
    f2+8304bcf3+1258c2f3+8248bdf3-6374cdf3-6552d2f3+9252bef3+12153cef3-15012d\
    ef3-13865e2f3+6826bf4+2029cf4-4303df4+1572ef4-3519f5-11980c4g+9748bd3g+86\
    60cd3g+15466d4g+6444c3eg-4232bcdeg+4568c2deg-14626bd2eg-786cd2eg-3625d3eg\
    +1987bce2g+4552c2e2g-11880bde2g-15172cde2g-15100d2e2g+1684be3g-2235ce3g-2\
    457de3g+7861e4g-3828c3fg+11118bcdfg+2913c2dfg+6653bd2fg-11706cd2fg-9675d3\
    fg+13393bcefg-8499c2efg+4752bdefg-3700cdefg+13113d2efg-2959be2fg-5611ce2f\
    g+15743de2fg+7880e3fg+8042bcf2g+15356c2f2g+4811bdf2g-8874cdf2g+12656d2f2g\
    +941bef2g+7466cef2g+10054def2g-9344e2f2g-10679bf3g-14371cf3g-12656df3g-51\
    04ef3g+3433f4g-5370c3g2-3464bcdg2-8570c2dg2+6077bd2g2+3404cd2g2-6569d3g2-\
    9558bceg2+15307c2eg2-14132bdeg2-10957cdeg2-8341d2eg2+8321be2g2+1980ce2g2-\
    8582de2g2+8677e3g2-11191bcfg2+10521c2fg2+9012bdfg2-13634cdfg2-3808d2fg2+3\
    919befg2+6111cefg2+10508defg2+2202e2fg2+3217bf2g2+10056cf2g2+2240df2g2-14\
    043ef2g2-9755f3g2+5900bcg3+15877c2g3+5228bdg3-3217cdg3-4407d2g3-6835beg3+\
    9399ceg3+882deg3-8551e2g3+12586bfg3-11500cfg3+8923dfg3+7550efg3-6574f2g3-\
    14049bg4+13640cg4+15435dg4+4111eg4-15602fg4-10030g5
  ↦ j[8]=c2e2f-8933cde2f-4268d2e2f-9066be3f-1129ce3f-2379de3f-14284e4f-13185c\
    3f2+7955bcdf2-7656c2df2-5995bd2f2+9705cd2f2-7658d3f2-10633bcef2+2156c2ef2\
    +3772bdef2+511cdef2-14468d2ef2-2189be2f2+1185ce2f2-1262de2f2-8051e3f2+729\
    9bcf3-11885c2f3-4273bdf3-7906cdf3+6861d2f3+11955bef3+4630cef3+2964def3+71\
    31e2f3+5282bf4-2891cf4+14815df4-8785ef4+13656f5-10635c4g+8702bd3g+9201cd3\
    g-1658d4g-54c3eg+10048bcdeg-518c2deg-12380bd2eg+10258cd2eg-4566d3eg+12844\
    bce2g+6270c2e2g+3597bde2g-15397cde2g-6656d2e2g-3405be3g-6121ce3g-3569de3g\
    +8570e4g+4050c3fg+10429bcdfg-1404c2dfg+2774bd2fg+925cd2fg-13141d3fg+4541b\
    cefg-163c2efg+239bdefg-11837cdefg+20d2efg-7079be2fg+7908ce2fg-13967de2fg-\
    10196e3fg+6866bcf2g-8561c2f2g+8636bdf2g+5525cdf2g+15012d2f2g-1791bef2g+41\
    38cef2g-9241def2g-381e2f2g-5532bf3g+3724cf3g+276df3g-14658ef3g+1614f4g-17\
    53c3g2+9827bcdg2+8273c2dg2+7169bd2g2+8052cd2g2+14188d3g2-14275bceg2+9744c\
    2eg2-5918bdeg2+6688cdeg2+1725d2eg2-5479be2g2+1943ce2g2-9880de2g2-6817e3g2\
    +3043bcfg2-5873c2fg2+13324bdfg2+5776cdfg2+9264d2fg2-12370befg2-7265cefg2-\
    5496defg2+6203e2fg2-9649bf2g2+11569cf2g2+8242df2g2+15447ef2g2-697f3g2-624\
    3bcg3-3772c2g3-4440bdg3-6727cdg3-8078d2g3-13750beg3+8809ceg3-635deg3+1410\
    1e2g3+9560bfg3-4795cfg3-9438dfg3+14729efg3+3215f2g3-13395bg4-8502cg4-4584\
    dg4+13498eg4+10895fg4-13695g5
  ↦ j[9]=bce2f-5850cde2f-14924d2e2f-5199be3f+6262ce3f+13178de3f-12690e4f+4303\
    c3f2-13588bcdf2+5315c2df2+3114bd2f2+4653cd2f2-13375d3f2+14771bcef2-5098c2\
    ef2-8643bdef2+15016cdef2-8753d2ef2+3563be2f2-14177ce2f2+1073de2f2+12082e3\
    f2+5309bcf3+13973c2f3+52bdf3+14778cdf3+1081d2f3+6615bef3-11602cef3-9386de\
    f3-11906e2f3-3517bf4-11308cf4+12379df4+3476ef4-7554f5-6239c4g+2149bd3g-35\
    13cd3g-7336d4g+9939c3eg-4571bcdeg-3543c2deg-13157bd2eg+9470cd2eg-1405d3eg\
    +337bce2g-9511c2e2g-10448bde2g-9806cde2g+4213d2e2g+9120be3g+15932ce3g-316\
    de3g+8918e4g-12680c3fg+5134bcdfg+4112c2dfg-146bd2fg-12700cd2fg+5797d3fg+7\
    103bcefg-5411c2efg-14152bdefg-15886cdefg-4202d2efg-15679be2fg-15852ce2fg+\
    15826de2fg+12803e3fg-5985bcf2g+538c2f2g+10066bdf2g+10525cdf2g+12625d2f2g+\
    7709bef2g-7974cef2g+5828def2g-12790e2f2g-5110bf3g+12772cf3g-7500df3g-9657\
    ef3g+14399f4g+2939c3g2+7561bcdg2-10112c2dg2+3000bd2g2+8776cd2g2+3230d3g2+\
    11408bceg2-1850c2eg2+6286bdeg2+11602cdeg2-3149d2eg2+2836be2g2+1159ce2g2+8\
    709de2g2-14272e3g2-14973bcfg2+13697c2fg2-9111bdfg2-531cdfg2-14085d2fg2-13\
    767befg2-11798cefg2+3787defg2-2863e2fg2-330bf2g2+3615cf2g2+5110df2g2-1677\
    ef2g2+14334f3g2-11385bcg3+13784c2g3-3550bdg3-13346cdg3+5924d2g3-8214beg3-\
   857ceg3+10933deg3+1884e2g3-9164bfg3+14566cfg3+15888dfg3-14984efg3-4908f2g\
   3+549bg4-5018cg4-14211dg4-3934eg4-9619fg4+8955g5
  ↦ j[10]=d3ef-7055cde2f-13905d2e2f-14726be3f+1413ce3f-5321de3f+12358e4f-2367\
   c3f2-3131bcdf2-5376c2df2-10591bd2f2+11940cd2f2+814d3f2+5954bcef2-15166c2e\
   f2+276bdef2+10026cdef2+9353d2ef2-789be2f2+15416ce2f2+5879de2f2+8142e3f2+6\
   320bcf3-9096c2f3+15309bdf3-12007cdf3+13192d2f3-6599bef3+5061cef3-11735def\
   3-4432e2f3+2800bf4+11608cf4+14371df4+4336ef4+6175f5-12338c4g-427bd3g+440c\
   d3g-2894d4g-3245c3eg+11723bcdeg+1422c2deg-15894bd2eg+6243cd2eg-14173d3eg-\
   11320bce2g-6641c2e2g+2151bde2g+5113cde2g+6927d2e2g+4071be3g+3666ce3g-1117\
   8de3g-8976e4g+14022c3fg+12bcdfg+1415c2dfg+7112bd2fg+2779cd2fg+15979d3fg-1\
   4796bcefg-1094c2efg+11318bdefg+4480cdefg+11844d2efg+15027be2fg-6246ce2fg-\
   15086de2fg+4133e3fg-6944bcf2g-2808c2f2g+12882bdf2g+3044cdf2g-7492d2f2g+22\
   69bef2g-170cef2g+9993def2g-5487e2f2g+4265bf3g-6238cf3g-3112df3g+2246ef3g+\
   1915f4g-13128c3g2-6782bcdg2-7033c2dg2-15950bd2g2-1044cd2g2-5170d3g2-4993b\
   ceg2-10654c2eg2+2721bdeg2-3576cdeg2-8961d2eg2+5350be2g2-11634ce2g2+13905d\
   e2g2-1896e3g2+11196bcfg2-5691c2fg2+7519bdfg2-2273cdfg2+52d2fg2+12285befg2\
   -15521cefg2+13030defg2+10901e2fg2+4901bf2g2+15618cf2g2-15425df2g2+12850ef\
   2g2+5990f3g2-11735bcg3+8090c2g3+10906bdg3-3034cdg3-912d2g3-8930beg3-4536c\
   eg3-10490deg3+13543e2g3+4290bfg3+15367cfg3+9861dfg3-7916efg3-6554f2g3+495\
   3bg4-15035cg4+6718dg4-6828eg4+11031fg4-11753g5
  ↦ j[11]=cd2ef+14948cde2f+1906d2e2f+3503be3f+1452ce3f+3425de3f-9473e4f+3512c\
   3f2-1770bcdf2-6597c2df2-2907bd2f2+1571cd2f2-1474d3f2+8029bcef2+14883c2ef2\
   +7094bdef2+2769cdef2-122d2ef2-11928be2f2+3654ce2f2+3868de2f2+5300e3f2-111\
   08bcf3-7680c2f3+14241bdf3+14211cdf3+9324d2f3-2846bef3+432cef3-4039def3+49\
   11e2f3-5624bf4+9397cf4+12542df4-1765ef4+878f5-2205c4g+5823bd3g-7671cd3g-3\
   558d4g-5100c3eg+12579bcdeg+11306c2deg-6942bd2eg-3172cd2eg+1410d3eg+2520bc\
   e2g+11554c2e2g-2858bde2g+7961cde2g+3682d2e2g+12759be3g-76ce3g+4742de3g+58\
   87e4g-2985c3fg+1414bcdfg-15593c2dfg+1140bd2fg+14543cd2fg+4460d3fg+1190bce\
   fg+1994c2efg-9849bdefg-12884cdefg+15708d2efg+14015be2fg-13014ce2fg-3195de\
   2fg+4979e3fg+8067bcf2g+10219c2f2g+616bdf2g+15192cdf2g-5903d2f2g-3156bef2g\
   -12171cef2g-966def2g-9268e2f2g+5151bf3g+3853cf3g-14717df3g+4583ef3g+3021f\
   4g+14180c3g2-4911bcdg2-12081c2dg2+1164bd2g2+13407cd2g2+9689d3g2+15079bceg\
   2-14506c2eg2+13946bdeg2-15021cdeg2-389d2eg2-1888be2g2-14633ce2g2-8740de2g\
   2+8525e3g2+12526bcfg2+12591c2fg2+4330bdfg2+6549cdfg2+7307d2fg2-1419befg2+\
   13248cefg2+5113defg2-322e2fg2-10953bf2g2+1230cf2g2+14770df2g2-13468ef2g2-\
   1460f3g2-11936bcg3+12108c2g3+1841bdg3-7103cdg3+14230d2g3+9641beg3-14328ce\
   g3-2000deg3-10541e2g3+14812bfg3+2610cfg3-15600dfg3-14710efg3+5356f2g3+146\
   24bg4-3745cg4-15554dg4+13088eg4-4484fg4-11760g5
  ↦ j[12]=bd2ef+9230cde2f-11199d2e2f+3103be3f-2270ce3f-6970de3f+11142e4f-7785\
   c3f2-155bcdf2+10348c2df2+3927bd2f2-6772cd2f2-9547d3f2-10568bcef2-10692c2e\
   f2-15735bdef2-10961cdef2+6694d2ef2-14107be2f2+6929ce2f2-4853de2f2+13714e3\
   f2-5899bcf3+6660c2f3-14559bdf3-2018cdf3-1946d2f3+2112bef3-9118cef3+9377de\
   f3+11330e2f3+15195bf4-14699cf4+14090df4+6140ef4-5672f5-7534c4g+7891bd3g-3\
   8cd3g+112d4g-8821c3eg+8314bcdeg+13196c2deg+7819bd2eg-2746cd2eg-11064d3eg-\
   12772bce2g-2903c2e2g+2895bde2g+1457cde2g+10483d2e2g+95be3g+4966ce3g-3109d\
   e3g+12955e4g+4311c3fg+7113bcdfg+396c2dfg-7950bd2fg+487cd2fg+4048d3fg+1476\
   7bcefg+8899c2efg-12754bdefg+1364cdefg+2781d2efg-9640be2fg+15540ce2fg-1151\
   5de2fg-985e3fg-13838bcf2g-1373c2f2g-31bdf2g+2189cdf2g+2633d2f2g-3332bef2g\
   -3056cef2g+12635def2g+3631e2f2g+11550bf3g-5473cf3g-12518df3g+10607ef3g-11\
   613f4g+9369c3g2-4902bcdg2+611c2dg2-1559bd2g2-6246cd2g2+3567d3g2+2811bceg2\
   +12311c2eg2-1663bdeg2-7263cdeg2+1911d2eg2-12791be2g2-7293ce2g2+13147de2g2\
   +12349e3g2-6864bcfg2-9645c2fg2-4654bdfg2-14968cdfg2-3335d2fg2+2634befg2-3\
   4cefg2-10443defg2+191e2fg2+7395bf2g2+148cf2g2+2553df2g2-14491ef2g2+14162f\
    3g2+1977bcg3-4857c2g3+1128bdg3-1136cdg3-2657d2g3-4923beg3+11388ceg3-10765\
    deg3+3747e2g3+629bfg3-4524cfg3+15627dfg3-13201efg3+2965f2g3-6666bg4-8441c\
    g4-15063dg4+8889eg4+486fg4-10153g5
  ↦ j[13]=c2def-14566cde2f-5384d2e2f-9581be3f-13148ce3f+6535de3f+12678e4f-851\
    7c3f2+9315bcdf2-12834c2df2+6136bd2f2+4727cd2f2-8656d3f2+5368bcef2+7034c2e\
    f2-8591bdef2+2288cdef2+2053d2ef2-4209be2f2-3386ce2f2+12519de2f2-7035e3f2-\
    11256bcf3-5726c2f3-8427bdf3+8767cdf3+8145d2f3-6419bef3+8166cef3-5959def3-\
    9158e2f3-3560bf4-9271cf4+12907df4-12140ef4+14921f5+10554c4g+15710bd3g+246\
    7cd3g-6506d4g-11518c3eg-15765bcdeg-2974c2deg-3836bd2eg+8225cd2eg+5516d3eg\
    +9685bce2g+10889c2e2g-7957bde2g+14678cde2g+12887d2e2g+8837be3g-7395ce3g+1\
    464de3g-5461e4g+11185c3fg-794bcdfg-2908c2dfg+907bd2fg-13098cd2fg+14889d3f\
    g+14684bcefg-179c2efg-11179bdefg-587cdefg-5377d2efg+7002be2fg-4211ce2fg-1\
    2076de2fg+1178e3fg-8160bcf2g+2744c2f2g-5077bdf2g-4056cdf2g-11822d2f2g+157\
    94bef2g+11200cef2g-4219def2g+8744e2f2g-6101bf3g+691cf3g+2836df3g-13989ef3\
    g-143f4g-3753c3g2-7345bcdg2-4271c2dg2+7693bd2g2+754cd2g2-1112d3g2+10670bc\
    eg2+14586c2eg2-14538bdeg2+9572cdeg2+1549d2eg2-4533be2g2-3093ce2g2+3100de2\
    g2+15237e3g2+13604bcfg2-14618c2fg2-13126bdfg2+10005cdfg2-3090d2fg2+2056be\
    fg2+1585cefg2+10072defg2-10576e2fg2-9036bf2g2+5461cf2g2+15279df2g2-10656e\
    f2g2-12524f3g2+6404bcg3-2805c2g3-4912bdg3-10921cdg3-4664d2g3-5990beg3+634\
    6ceg3-5383deg3-7749e2g3-14292bfg3+6929cfg3+11094dfg3-15280efg3-1058f2g3-1\
    3187bg4+8206cg4+13388dg4-1642eg4+11722fg4+1724g5
  ↦ j[14]=bcdef+2076cde2f+2789d2e2f+6285be3f+13607ce3f+6204de3f+11616e4f-829c\
    3f2+3834bcdf2-14613c2df2-702bd2f2+6848cd2f2+2569d3f2+4965bcef2-7492c2ef2+\
    543bdef2-12629cdef2-3320d2ef2-4498be2f2+14651ce2f2-994de2f2-4287e3f2+1406\
    7bcf3+13800c2f3-14148bdf3-10698cdf3-6795d2f3+10258bef3+15744cef3-13294def\
    3-6108e2f3+7421bf4-10860cf4+10913df4+7320ef4+12109f5-10226c4g-9873bd3g-99\
    46cd3g+1439d4g+3248c3eg+9622bcdeg+6067c2deg-12232bd2eg-2566cd2eg+10419d3e\
    g-14322bce2g-15434c2e2g-7883bde2g-14822cde2g-9906d2e2g+13298be3g-15723ce3\
    g-15727de3g+11496e4g+13460c3fg+177bcdfg+13862c2dfg-12872bd2fg+11085cd2fg-\
    13541d3fg+9380bcefg+5400c2efg+6723bdefg+15499cdefg-13895d2efg-12007be2fg+\
    11227ce2fg+10049de2fg+9042e3fg+3313bcf2g-2432c2f2g+15447bdf2g+15668cdf2g+\
    7598d2f2g+9979bef2g-13756cef2g-2251def2g-15924e2f2g-11611bf3g-11444cf3g+1\
    5626df3g-42ef3g+11298f4g-4098c3g2+5128bcdg2-9895c2dg2-15439bd2g2+5834cd2g\
    2+3846d3g2-2916bceg2-4955c2eg2-12709bdeg2-7996cdeg2+8882d2eg2-6854be2g2-1\
    3400ce2g2-13693de2g2-14267e3g2+10815bcfg2-15102c2fg2-10117bdfg2-12966cdfg\
    2+13250d2fg2+2770befg2-6371cefg2-7124defg2-13319e2fg2-14435bf2g2+8923cf2g\
    2-15222df2g2+6441ef2g2-3658f3g2+12575bcg3-12956c2g3-405bdg3+15313cdg3+152\
    09d2g3-3849beg3-13879ceg3-6296deg3+8899e2g3-741bfg3-7525cfg3+6567dfg3-991\
    4efg3-4256f2g3-10991bg4-9858cg4-11848dg4+3886eg4-11267fg4+3631g5
  ↦ j[15]=c3ef+3627cde2f-3024d2e2f+12379be3f+6006ce3f+8813de3f+1333e4f-9008c3\
    f2-15048bcdf2+13702c2df2+444bd2f2+3409cd2f2-3998d3f2+11388bcef2-8631c2ef2\
    +10732bdef2-2107cdef2+9540d2ef2-7777be2f2+3283ce2f2+14877de2f2+12070e3f2-\
    6697bcf3-14979c2f3-15112bdf3-4347cdf3+2232d2f3+8787bef3-8609cef3+497def3+\
    1448e2f3+968bf4-10619cf4+10064df4-5428ef4-12459f5+2201c4g+9840bd3g+3772cd\
    3g+15550d4g-1196c3eg+5132bcdeg+12325c2deg+14615bd2eg+2401cd2eg-14892d3eg-\
    10894bce2g+13600c2e2g+10482bde2g+10982cde2g-15077d2e2g-15318be3g-9232ce3g\
    -14493de3g+7447e4g-12128c3fg-8318bcdfg+7366c2dfg-1131bd2fg+1309cd2fg+1127\
    6d3fg+13915bcefg-14668c2efg+14462bdefg-10844cdefg+4992d2efg-3153be2fg+409\
    2ce2fg+1594de2fg-14441e3fg-1022bcf2g+5974c2f2g-13587bdf2g-cdf2g-8575d2f2g\
    +14676bef2g+10697cef2g-7492def2g-13282e2f2g+6820bf3g-10932cf3g+3872df3g-1\
    5753ef3g+14248f4g+5018c3g2+4301bcdg2+13799c2dg2-7020bd2g2-13806cd2g2+9843\
    d3g2-9041bceg2+14886c2eg2+1068bdeg2+12615cdeg2+9832d2eg2-991be2g2+9254ce2\
    g2-10752de2g2-6367e3g2+12175bcfg2+4608c2fg2+10123bdfg2-10249cdfg2+1612d2f\
    g2-3299befg2+11732cefg2+915defg2+2423e2fg2-6019bf2g2+6385cf2g2-8708df2g2+\
    11446ef2g2+8934f3g2-15994bcg3+10661c2g3+3320bdg3+5494cdg3+9577d2g3+6558be\
    g3-1113ceg3+6863deg3-15749e2g3+12538bfg3-14385cfg3-2589dfg3-10620efg3+918\
    f2g3-12299bg4-8962cg4-4070dg4-11387eg4-7448fg4-8995g5
 ↦ j[16]=d4f+8527cde2f-6944d2e2f-1640be3f-1643ce3f-12214de3f+14973e4f+8163c3\
    f2+9263bcdf2+6914c2df2+8441bd2f2+13263cd2f2-4202d3f2-4237bcef2-11247c2ef2\
    -8388bdef2-12422cdef2+8737d2ef2+1065be2f2+5407ce2f2+176de2f2+10568e3f2+51\
    58bcf3-5121c2f3+930bdf3+13996cdf3+1460d2f3+107bef3-751cef3-3318def3+15420\
    e2f3-460bf4-6673cf4-9697df4-4887ef4+12404f5-7635c4g-5295bd3g+10838cd3g-83\
    02d4g+14948c3eg+11628bcdeg+3448c2deg-9842bd2eg-3496cd2eg-4665d3eg+617bce2\
    g-5143c2e2g-1945bde2g-14900cde2g-4843d2e2g-4928be3g+14463ce3g-15905de3g-7\
    21e4g-10267c3fg-3305bcdfg-14001c2dfg-2775bd2fg-3456cd2fg-14717d3fg-10062b\
    cefg+2266c2efg-840bdefg-14845cdefg+3433d2efg+7762be2fg-68ce2fg+1805de2fg+\
    7728e3fg-5529bcf2g-2915c2f2g+4933bdf2g-9260cdf2g-14153d2f2g-7644bef2g-589\
    4cef2g+4594def2g+7147e2f2g-11389bf3g-5882cf3g+12633df3g-8236ef3g+8531f4g-\
    11656c3g2+11664bcdg2-206c2dg2-7569bd2g2-6219cd2g2-12331d3g2+10400bceg2+14\
    657c2eg2-6367bdeg2+3522cdeg2+6777d2eg2+8316be2g2-1739ce2g2+2339de2g2-2447\
    e3g2+11936bcfg2+3167c2fg2+5505bdfg2+11351cdfg2+9612d2fg2-1410befg2+13285c\
    efg2+1703defg2+2901e2fg2+5027bf2g2+8690cf2g2+5005df2g2-14286ef2g2-15398f3\
    g2-13571bcg3-2423c2g3-14174bdg3-667cdg3-13250d2g3-9459beg3+6763ceg3-5062d\
    eg3+3982e2g3+13043bfg3-14006cfg3-4859dfg3-15020efg3+15855f2g3+15034bg4+39\
    0cg4+10540dg4+11218eg4-610fg4-7001g5
 ↦ j[17]=cd3f+12751cde2f-10231d2e2f+6781be3f-8331ce3f-15726de3f+14382e4f-974\
    1c3f2-15571bcdf2-15278c2df2-304bd2f2+23cd2f2+9368d3f2+1773bcef2+8236c2ef2\
    +10749bdef2-4782cdef2+6891d2ef2+9048be2f2+5927ce2f2-13042de2f2+107e3f2+54\
    67bcf3+11899c2f3+12883bdf3-8014cdf3+2288d2f3-11561bef3+10362cef3-1166def3\
    +3306e2f3+11146bf4-9480cf4+14769df4-14494ef4+12536f5-13565c4g-4244bd3g-14\
    385cd3g-9531d4g-221c3eg+6480bcdeg-6836c2deg+3118bd2eg-2608cd2eg+14897d3eg\
    -14648bce2g-11052c2e2g+6907bde2g-9644cde2g+6045d2e2g-14541be3g-8370ce3g+2\
    005de3g-3178e4g+7171c3fg+5572bcdfg-9609c2dfg-1125bd2fg-11924cd2fg+7694d3f\
    g+9608bcefg+2855c2efg-13575bdefg+15546cdefg+6426d2efg+2759be2fg-772ce2fg+\
    2370de2fg+5566e3fg-6295bcf2g+12068c2f2g+8646bdf2g+5809cdf2g-174d2f2g+1094\
    8bef2g+11197cef2g-228def2g-9634e2f2g-12121bf3g-216cf3g-10116df3g+14812ef3\
    g-8460f4g+3898c3g2+690bcdg2-15659c2dg2+8753bd2g2+5451cd2g2+12546d3g2+1124\
    bceg2+7639c2eg2+110bdeg2+5153cdeg2-14484d2eg2-14629be2g2-2758ce2g2-4287de\
    2g2+12331e3g2+10265bcfg2+4679c2fg2-7570bdfg2-15230cdfg2+2741d2fg2-12201be\
    fg2-6624cefg2-13652defg2+14080e2fg2+685bf2g2+15713cf2g2+4896df2g2+13188ef\
    2g2+301f3g2-12294bcg3-9070c2g3-5151bdg3+5120cdg3+5227d2g3-6730beg3+15802c\
    eg3+15318deg3-6876e2g3+637bfg3-15229cfg3-14dfg3-9692efg3-13009f2g3-13592b\
    g4-5686cg4-3976dg4-10863eg4+7333fg4+9194g5
 ↦ j[18]=bd3f-11270cde2f-849d2e2f+14644be3f-15900ce3f-9122de3f+3754e4f+6414c\
    3f2+304bcdf2+6538c2df2-13424bd2f2-2662cd2f2-12151d3f2+10029bcef2+10393c2e\
    f2+13169bdef2+2415cdef2+13291d2ef2-9773be2f2+8331ce2f2-1263de2f2+7006e3f2\
    +9868bcf3+12073c2f3-4480bdf3-5259cdf3+3261d2f3+1645bef3-12796cef3-3555def\
    3-14620e2f3-15287bf4-4462cf4+9374df4-11818ef4+14837f5-7008c4g+15172bd3g-7\
    706cd3g+15185d4g+5551c3eg-6170bcdeg+12776c2deg+13017bd2eg-4800cd2eg-15788\
    d3eg-15124bce2g-4422c2e2g-14653bde2g-15615cde2g-5297d2e2g+14688be3g-8564c\
    e3g+4133de3g-15515e4g-12178c3fg+7754bcdfg+1583c2dfg-177bd2fg+5377cd2fg-14\
    118d3fg-12266bcefg-757c2efg-15178bdefg-2273cdefg-15658d2efg-22be2fg+13287\
    ce2fg-10107de2fg-6502e3fg-12510bcf2g-9618c2f2g+15033bdf2g-14220cdf2g-8826\
    d2f2g+12853bef2g+5637cef2g+14513def2g+1630e2f2g-13893bf3g+5089cf3g-10811d\
    f3g+5967ef3g-13145f4g-1269c3g2+9447bcdg2-8205c2dg2-10925bd2g2+7529cd2g2+7\
    420d3g2+12752bceg2+5381c2eg2+5968bdeg2+13090cdeg2+9964d2eg2-4197be2g2-506\
    4ce2g2+13616de2g2-2816e3g2+8245bcfg2-6363c2fg2+10611bdfg2-7411cdfg2+2033d\
    2fg2+9793befg2-8898cefg2-10153defg2+3116e2fg2+15106bf2g2-4448cf2g2-643df2\
    g2+13878ef2g2+8836f3g2+14638bcg3+13808c2g3-10593bdg3+5424cdg3+3312d2g3-14\
    237beg3+12077ceg3+10222deg3+7961e2g3-10658bfg3+6904cfg3+11268dfg3-14675ef\
    g3-288f2g3+833bg4-9142cg4-5311dg4-5967eg4+1214fg4-10166g5
 ↦ j[19]=c4f+7081cde2f-2263d2e2f-13205be3f-14299ce3f-9841de3f-661e4f+11146c3\
    f2-6863bcdf2-15828c2df2+11190bd2f2+987cd2f2-9885d3f2-9188bcef2-5751c2ef2+\
    556bdef2+8750cdef2+15750d2ef2+10748be2f2+3937ce2f2+6173de2f2+13821e3f2+99\
    50bcf3-14776c2f3+10400bdf3+3142cdf3-13339d2f3-7431bef3-15633cef3+14099def\
    3-6392e2f3+627bf4+14590cf4+13498df4+2353ef4+3157f5-6971c4g-12399bd3g+5491\
    cd3g+5865d4g+5414c3eg+13166bcdeg-12459c2deg-14740bd2eg-14403cd2eg-8136d3e\
    g+7208bce2g-11722c2e2g-12508bde2g+7384cde2g-2800d2e2g-2088be3g-13932ce3g+\
    5469de3g-8094e4g+14307c3fg+9949bcdfg-5064c2dfg-1671bd2fg-8907cd2fg+10899d\
    3fg+14194bcefg+15422c2efg+12690bdefg-1692cdefg-14468d2efg+2326be2fg-2756c\
    e2fg-9154de2fg+3546e3fg-15671bcf2g+1496c2f2g+12536bdf2g+3254cdf2g-3701d2f\
    2g-2339bef2g+2603cef2g-6329def2g+10173e2f2g+6657bf3g-7771cf3g+1837df3g+31\
    14ef3g-15629f4g-7752c3g2+1556bcdg2-5945c2dg2-11332bd2g2+15016cd2g2+1001d3\
    g2-10064bceg2+18c2eg2+7229bdeg2+872cdeg2-14117d2eg2-5839be2g2+15293ce2g2-\
    7240de2g2-8979e3g2+8016bcfg2-14237c2fg2+8596bdfg2+1327cdfg2-9831d2fg2-108\
    18befg2+2865cefg2+2576defg2-12988e2fg2-13310bf2g2+1433cf2g2-5777df2g2+108\
    30ef2g2+7268f3g2+7812bcg3+10741c2g3-9747bdg3+3784cdg3-15249d2g3+7977beg3+\
    10192ceg3+8538deg3-12194e2g3-266bfg3+7744cfg3+8052dfg3-5432efg3+1690f2g3-\
    3904bg4+14910cg4-14503dg4+8746eg4+8503fg4+8739g5
 ↦ j[20]=e5+5026cde2f+4629d2e2f-4847be3f-4525ce3f-5882de3f+10459e4f-3003c3f2\
    -935bcdf2-4215c2df2-14019bd2f2+7020cd2f2+8118d3f2+3202bcef2-1307c2ef2+103\
    98bdef2+8723cdef2+3814d2ef2-4312be2f2-15786ce2f2+1759de2f2-11483e3f2+9563\
    bcf3+5954c2f3+12503bdf3-1569cdf3-13500d2f3-10029bef3-13581cef3-9962def3-4\
    235e2f3+15712bf4-14611cf4-5563df4+5801ef4+13465f5-15477c4g+5157bd3g-11358\
    cd3g+4542d4g+2291c3eg+15311bcdeg+11730c2deg-13588bd2eg-2062cd2eg-15638d3e\
    g-10017bce2g-2628c2e2g+14351bde2g-10695cde2g-9691d2e2g-4270be3g+13424ce3g\
    +11438de3g-11133e4g+1144c3fg+11548bcdfg+7741c2dfg-8428bd2fg-12060cd2fg-73\
    7d3fg+9577bcefg+10965c2efg-5060bdefg-14559cdefg+1354d2efg+1811be2fg-789ce\
    2fg-9567de2fg+2328e3fg-6891bcf2g+4667c2f2g+4502bdf2g-11869cdf2g-3774d2f2g\
    +7528bef2g-14667cef2g+1274def2g+15266e2f2g-3550bf3g-9587cf3g+6568df3g+140\
    54ef3g+7880f4g+4088c3g2+13371bcdg2-11585c2dg2-9251bd2g2+5016cd2g2-9351d3g\
    2+11682bceg2+7777c2eg2+2793bdeg2+5260cdeg2-1282d2eg2+15866be2g2+4053ce2g2\
    +9580de2g2-4383e3g2-11856bcfg2+13051c2fg2-2938bdfg2+5881cdfg2+1752d2fg2+1\
    202befg2-12009cefg2+2854defg2-1615e2fg2-4421bf2g2-7716cf2g2+10832df2g2-13\
    949ef2g2+2651f3g2-13111bcg3-3116c2g3-12565bdg3+15719cdg3+6676d2g3+11954be\
    g3+8524ceg3-6812deg3+6155e2g3-8017bfg3-2397cfg3+886dfg3-13087efg3+3970f2g\
    3+2077bg4-11262cg4+4737dg4-5781eg4-11435fg4+6390g5
 ↦ j[21]=de4+15589cde2f+746d2e2f-9942be3f+10908ce3f+1369de3f+11442e4f+13135c\
    3f2-6766bcdf2-13605c2df2+12873bd2f2+1012cd2f2+10486d3f2-3179bcef2+7503c2e\
    f2+14646bdef2-12157cdef2-7864d2ef2-3564be2f2-12748ce2f2-7485de2f2+2260e3f\
    2+2905bcf3-8587c2f3-5315bdf3-13135cdf3+9103d2f3+5131bef3+15125cef3+13102d\
    ef3-6295e2f3-2254bf4+1375cf4+2333df4+9497ef4-4205f5+15667c4g+10351bd3g-55\
    51cd3g-6567d4g-2011c3eg+4194bcdeg+7704c2deg-10345bd2eg-8554cd2eg-294d3eg-\
    9115bce2g+10113c2e2g-3125bde2g+4228cde2g+6435d2e2g-4641be3g+4876ce3g+4992\
    de3g-14904e4g-5290c3fg+14971bcdfg-2404c2dfg+4574bd2fg+1403cd2fg-14166d3fg\
    -9919bcefg+5782c2efg+4982bdefg-7792cdefg+399d2efg+10019be2fg+11327ce2fg+9\
    471de2fg-9998e3fg-2324bcf2g-7169c2f2g+8003bdf2g+9069cdf2g+14993d2f2g-1409\
    5bef2g+6329cef2g+11853def2g-8864e2f2g+15253bf3g+8441cf3g+5180df3g+15279ef\
    3g+1006f4g-14978c3g2+4238bcdg2+11482c2dg2-8852bd2g2-2959cd2g2+3610d3g2-68\
    24bceg2+6546c2eg2+11799bdeg2-15233cdeg2+8346d2eg2-14555be2g2-4322ce2g2+76\
    27de2g2+297e3g2-14244bcfg2-6056c2fg2-9177bdfg2+3186cdfg2-1507d2fg2+8402be\
    fg2+9527cefg2+14915defg2+13814e2fg2+6936bf2g2-7019cf2g2-2673df2g2+8525ef2\
    g2+2978f3g2+385bcg3-2867c2g3-13784bdg3+1239cdg3-15302d2g3+2458beg3-15819c\
    eg3-6098deg3+11055e2g3-3351bfg3-2456cfg3+992dfg3+7911efg3-1813f2g3+13251b\
    g4+12965cg4+15010dg4-10451eg4-14837fg4+12489g5
 ↦ j[22]=ce4-967cde2f+308d2e2f+12438be3f+7025ce3f+4783de3f+9961e4f+8422c3f2+\
    7001bcdf2-5399c2df2-10125bd2f2-6354cd2f2+13298d3f2-6639bcef2-11030c2ef2+2\
    520bdef2+9197cdef2-1394d2ef2-8753be2f2-13927ce2f2+14071de2f2-2620e3f2-588\
    1bcf3-15951c2f3-9186bdf3+15124cdf3+13165d2f3-5863bef3+7968cef3-7136def3-1\
    2976e2f3+4980bf4-4050cf4+534df4-3025ef4-7399f5-1784c4g-1736bd3g+1233cd3g+\
    5640d4g+9944c3eg+580bcdeg+4409c2deg+2482bd2eg+8512cd2eg+9245d3eg+780bce2g\
    +8384c2e2g+4714bde2g-7144cde2g+1554d2e2g-9461be3g+14323ce3g+10204de3g-931\
    9e4g-11676c3fg-13233bcdfg+10224c2dfg+2001bd2fg-1304cd2fg+2619d3fg+8932bce\
    fg-7537c2efg-3417bdefg+9740cdefg-13694d2efg+10589be2fg-6576ce2fg+203de2fg\
    -4919e3fg-8743bcf2g-5200c2f2g-6085bdf2g+2713cdf2g-9093d2f2g-2313bef2g+111\
    22cef2g-15730def2g-3131e2f2g-4661bf3g+4071cf3g+11922df3g-9534ef3g-2432f4g\
    -15206c3g2-12516bcdg2-7121c2dg2-9180bd2g2+9206cd2g2+15832d3g2-2567bceg2+2\
    204c2eg2+10266bdeg2+15602cdeg2-3957d2eg2-672be2g2+12019ce2g2-2679de2g2-15\
    485e3g2-9024bcfg2+7941c2fg2+3932bdfg2+6592cdfg2-101d2fg2+8370befg2+11163c\
    efg2-15638defg2+3234e2fg2-5000bf2g2-6712cf2g2-8627df2g2+15097ef2g2-12723f\
    3g2+11358bcg3-9260c2g3-14791bdg3+14176cdg3-11437d2g3-5737beg3+14117ceg3-1\
    391deg3-9096e2g3+3551bfg3+15222cfg3+14291dfg3-5158efg3+359f2g3-2917bg4+14\
    28cg4+10969dg4+2962eg4-5422fg4-7013g5
 ↦ j[23]=be4+13983cde2f+13096d2e2f-15942be3f-2263ce3f+1792de3f+11291e4f-1418\
    9c3f2-5132bcdf2-10030c2df2-4629bd2f2-6716cd2f2-8245d3f2-10787bcef2+5472c2\
    ef2-4839bdef2-10320cdef2-13480d2ef2+3056be2f2-5718ce2f2+5453de2f2+2440e3f\
    2-4454bcf3-10209c2f3+4560bdf3+5808cdf3+9557d2f3-15477bef3+508cef3+8635def\
    3-13431e2f3-15470bf4+12152cf4-1109df4-14459ef4-7499f5+6864c4g-9874bd3g-14\
    217cd3g-8784d4g+7002c3eg+4295bcdeg+5194c2deg+4013bd2eg-6246cd2eg-3635d3eg\
    -10398bce2g+9268c2e2g+6916bde2g-13809cde2g+10792d2e2g+12747be3g+7366ce3g+\
    5522de3g-12320e4g+13159c3fg-4808bcdfg+3242c2dfg+6817bd2fg-1704cd2fg-2354d\
    3fg+2194bcefg-6622c2efg+12947bdefg+15246cdefg+4888d2efg-14907be2fg+10412c\
    e2fg+11938de2fg-7542e3fg-15949bcf2g+7019c2f2g+1065bdf2g-11042cdf2g+8420d2\
    f2g-2947bef2g+10239cef2g-6103def2g+7514e2f2g+5564bf3g-3497cf3g-7497df3g+1\
    5098ef3g-11726f4g+3441c3g2+14798bcdg2-2363c2dg2-8955bd2g2+1560cd2g2-13163\
    d3g2-3378bceg2+14223c2eg2-6579bdeg2+10976cdeg2+4716d2eg2-9705be2g2+10150c\
    e2g2+4710de2g2-9318e3g2-1477bcfg2-1881c2fg2-10977bdfg2+10578cdfg2-6757d2f\
    g2-14861befg2-10354cefg2+2243defg2-3963e2fg2+5009bf2g2-5028cf2g2+3155df2g\
    2+12560ef2g2-15123f3g2-2203bcg3+9775c2g3-14037bdg3-15944cdg3-7281d2g3+442\
    5beg3-15925ceg3+9030deg3+9555e2g3-13756bfg3-8090cfg3-3359dfg3-15604efg3+5\
    671f2g3+5122bg4+2013cg4-6155dg4-3754eg4+10617fg4+4485g5
 ↦ j[24]=d2e3+9549cde2f-15285d2e2f-9666be3f+8890ce3f-10522de3f+14500e4f-1191\
    5c3f2-1237bcdf2+11555c2df2+3800bd2f2-12569cd2f2+13974d3f2+1792bcef2-13627\
    c2ef2-15087bdef2+9194cdef2+11032d2ef2-4399be2f2+15615ce2f2-2987de2f2+1217\
    1e3f2+1328bcf3+3319c2f3-11578bdf3+9993cdf3+2328d2f3-1513bef3+457cef3-1128\
    2def3+14682e2f3+3132bf4+11210cf4+11956df4+15331ef4+10764f5-3113c4g-1013bd\
    3g-14430cd3g-12351d4g-4741c3eg+4498bcdeg+2052c2deg+14139bd2eg+4244cd2eg-4\
    141d3eg+4593bce2g-9951c2e2g+8375bde2g+7118cde2g-15674d2e2g-13391be3g-1167\
    9ce3g-2301de3g-7173e4g+11375c3fg-4256bcdfg-14861c2dfg-13707bd2fg+10660cd2\
    fg+6813d3fg+10478bcefg-5786c2efg-5398bdefg+2654cdefg+14920d2efg+11385be2f\
    g-10450ce2fg-4813de2fg-10174e3fg-4377bcf2g-6484c2f2g-6185bdf2g+7944cdf2g-\
    4759d2f2g-14378bef2g+6023cef2g+4211def2g+9438e2f2g-7207bf3g-4414cf3g-8287\
    df3g+6795ef3g+12380f4g-14487c3g2+15570bcdg2+11562c2dg2+7296bd2g2+1364cd2g\
    2-5405d3g2+14801bceg2+1230c2eg2-1205bdeg2-7178cdeg2+5831d2eg2+12692be2g2+\
    11140ce2g2+13159de2g2-11933e3g2-5720bcfg2-14632c2fg2-517bdfg2-4539cdfg2-1\
    3004d2fg2-2091befg2-10736cefg2-1532defg2-15953e2fg2+915bf2g2+3186cf2g2-11\
    466df2g2-12409ef2g2-841f3g2-13062bcg3+11216c2g3+5063bdg3-5842cdg3+6209d2g\
    3+10307beg3+365ceg3+7890deg3+13536e2g3-11394bfg3-12492cfg3-12686dfg3+9332\
    efg3-12777f2g3-14786bg4+8723cg4-9595dg4-1176eg4+11213fg4+3348g5
 ↦ j[25]=cde3+1085cde2f-13995d2e2f-2128be3f+11777ce3f+9694de3f+14679e4f+7595\
    c3f2+5971bcdf2+3166c2df2+2125bd2f2-7752cd2f2+5914d3f2-1994bcef2+4255c2ef2\
    +10312bdef2+8341cdef2+7068d2ef2-3629be2f2+10220ce2f2-8411de2f2-5541e3f2+3\
    307bcf3-11044c2f3+3397bdf3-11939cdf3-14415d2f3+1924bef3+1858cef3-3972def3\
    -3745e2f3+13607bf4+3970cf4+9283df4+9838ef4-9925f5-12491c4g+6224bd3g+4906c\
    d3g+4875d4g+1932c3eg+12361bcdeg+8006c2deg+3372bd2eg+1466cd2eg-7947d3eg+83\
    68bce2g-5298c2e2g-3816bde2g+6630cde2g+10337d2e2g+7222be3g-3926ce3g+5392de\
    3g+11096e4g-11972c3fg+733bcdfg-6618c2dfg-4284bd2fg-12736cd2fg+12386d3fg-5\
    185bcefg-7854c2efg+15301bdefg-8195cdefg-5785d2efg+15823be2fg-4415ce2fg+17\
    16de2fg+3532e3fg+5509bcf2g+4138c2f2g+1116bdf2g-7878cdf2g-12118d2f2g+5226b\
    ef2g-12723cef2g+1070def2g+4404e2f2g-15240bf3g+11743cf3g+4067df3g-5675ef3g\
    +8588f4g+13467c3g2-7012bcdg2+10504c2dg2+5021bd2g2-5405cd2g2+11130d3g2-668\
    6bceg2-2904c2eg2+3240bdeg2+12126cdeg2+2959d2eg2+13472be2g2-6963ce2g2-2811\
    de2g2-4373e3g2+4979bcfg2+10644c2fg2+14950bdfg2+4944cdfg2+13485d2fg2+7255b\
    efg2-10133cefg2-13007defg2-4942e2fg2-11991bf2g2+3083cf2g2+6706df2g2-5296e\
    f2g2+6302f3g2-6706bcg3+12531c2g3+11125bdg3-6314cdg3+13059d2g3+12866beg3-1\
    4921ceg3+13115deg3+11938e2g3+13142bfg3+15312cfg3+2068dfg3+11909efg3+9002f\
    2g3-11265bg4-1020cg4-2761dg4+12021eg4+961fg4+12938g5
 ↦ j[26]=bde3+13157cde2f-10459d2e2f-4981be3f-1980ce3f+11036de3f-4933e4f-1576\
    4c3f2+4392bcdf2-1453c2df2-13407bd2f2+2029cd2f2-7229d3f2-7872bcef2+11007c2\
    ef2-11494bdef2-10414cdef2-15681d2ef2-6807be2f2+1179ce2f2-6626de2f2+8748e3\
    f2+11736bcf3+10395c2f3+14219bdf3+12441cdf3+10261d2f3-15951bef3+11289cef3+\
    9022def3-8885e2f3-10829bf4+12403cf4+13262df4+1178ef4-15891f5-4187c4g-4226\
    bd3g-12920cd3g+5974d4g+10868c3eg+15873bcdeg-15204c2deg+6947bd2eg-9427cd2e\
    g+9431d3eg+15451bce2g+5293c2e2g+13384bde2g-10675cde2g+12810d2e2g-733be3g+\
    15311ce3g-3178de3g-1454e4g+3647c3fg+309bcdfg+309c2dfg-14410bd2fg-6418cd2f\
    g-14986d3fg-8575bcefg-2429c2efg-12847bdefg+8575cdefg-6981d2efg+9742be2fg+\
    8277ce2fg+6127de2fg+14280e3fg-8459bcf2g-5303c2f2g+3979bdf2g-488cdf2g+5325\
    d2f2g+3910bef2g-11261cef2g+3407def2g+4557e2f2g-9460bf3g-2646cf3g+369df3g+\
    90ef3g-11852f4g-15923c3g2+11619bcdg2-7126c2dg2-14018bd2g2-10380cd2g2+1326\
    4d3g2-10200bceg2+4379c2eg2-12038bdeg2+12436cdeg2-1483d2eg2-9841be2g2+9815\
    ce2g2+8612de2g2+8668e3g2-2379bcfg2-7052c2fg2-12938bdfg2-11993cdfg2-1860d2\
    fg2-7293befg2-13132cefg2-9014defg2+4555e2fg2-2503bf2g2-2809cf2g2-11752df2\
    g2+9498ef2g2-9890f3g2-11086bcg3+1042c2g3-11087bdg3-3450cdg3+636d2g3-7495b\
    eg3-3020ceg3-7429deg3+1573e2g3-8066bfg3-10695cfg3+12615dfg3+9422efg3-1003\
    1f2g3-579bg4-91cg4-2266dg4-3270eg4+1549fg4-10734g5
 ↦ j[27]=c2e3+15363cde2f-11044d2e2f+2827be3f+3324ce3f+8878de3f-7181e4f+7269c\
    3f2-14029bcdf2+10444c2df2-2834bd2f2+10961cd2f2+3404d3f2-11255bcef2-14712c\
    2ef2+987bdef2+6875cdef2-3355d2ef2-7707be2f2-11193ce2f2+10032de2f2+2000e3f\
    2+10102bcf3-10893c2f3-3919bdf3+27cdf3+1098d2f3-12355bef3-437cef3-5640def3\
    +8980e2f3+15281bf4-2350cf4+5288df4-3584ef4-2807f5-7544c4g+12189bd3g-3780c\
    d3g+2100d4g-10637c3eg-9650bcdeg-12833c2deg+13659bd2eg-7483cd2eg+1098d3eg-\
    6535bce2g-13976c2e2g-5653bde2g-7089cde2g+10798d2e2g+14599be3g+1052ce3g+34\
    11de3g-999e4g-10488c3fg+14104bcdfg-11615c2dfg+9295bd2fg+11328cd2fg-89d3fg\
    +6942bcefg-7795c2efg-12107bdefg+9428cdefg-1204d2efg+2256be2fg-229ce2fg+14\
    821de2fg-12094e3fg+3219bcf2g-12523c2f2g-13108bdf2g+5826cdf2g-5066d2f2g+66\
    57bef2g-4589cef2g-8442def2g-8505e2f2g-9071bf3g-7123cf3g+9569df3g-4462ef3g\
    -10756f4g+607c3g2+8351bcdg2+14170c2dg2-15116bd2g2+788cd2g2+5367d3g2+1059b\
    ceg2-1661c2eg2-3723bdeg2+12822cdeg2+15186d2eg2+6819be2g2+2835ce2g2+134de2\
    g2-9265e3g2-197bcfg2-1392c2fg2+3891bdfg2+5251cdfg2+2608d2fg2-5355befg2+33\
    86cefg2+12760defg2+11544e2fg2-14454bf2g2-13120cf2g2-4973df2g2+1442ef2g2-1\
    0617f3g2-12505bcg3-9740c2g3-15962bdg3+15028cdg3-8696d2g3+11962beg3-7093ce\
    g3+11335deg3+1461e2g3+5449bfg3+13386cfg3-12637dfg3+13230efg3-7509f2g3-611\
    6bg4+5081cg4-9639dg4-1200eg4-13324fg4+9584g5
 ↦ j[28]=bce3+10191cde2f-10041d2e2f-15432be3f+10550ce3f+9567de3f-5270e4f-153\
    4c3f2-14763bcdf2+8180c2df2-12765bd2f2+13485cd2f2-12564d3f2+5761bcef2-3836\
    c2ef2-13287bdef2-12874cdef2+10162d2ef2+5620be2f2+5498ce2f2-9484de2f2+4447\
    e3f2-334bcf3-5145c2f3-15436bdf3-15653cdf3+1388d2f3-4107bef3-9752cef3+1200\
    8def3+2831e2f3-9078bf4-10833cf4+12765df4-13262ef4+13020f5-15017c4g+14481b\
    d3g+2433cd3g-12754d4g-11047c3eg-3991bcdeg-4107c2deg+7430bd2eg+6850cd2eg-1\
    1203d3eg-14277bce2g-14364c2e2g+2908bde2g-1297cde2g+5856d2e2g+13796be3g+76\
    47ce3g+9519de3g+6008e4g-3259c3fg-13360bcdfg-5874c2dfg+14018bd2fg+7390cd2f\
    g-3676d3fg+14618bcefg-14292c2efg+9908bdefg+1147cdefg-15450d2efg+2477be2fg\
    +7726ce2fg+13475de2fg-15293e3fg+12700bcf2g-6816c2f2g+6465bdf2g+5057cdf2g-\
    6935d2f2g+241bef2g-15973cef2g-10699def2g-12895e2f2g+4323bf3g-11539cf3g-58\
    40df3g+10839ef3g+14576f4g-297c3g2-4427bcdg2+15078c2dg2-5695bd2g2-6516cd2g\
    2+14944d3g2-12858bceg2+13828c2eg2-12495bdeg2+13130cdeg2+10168d2eg2+3643be\
    2g2+12394ce2g2-835de2g2-6115e3g2-9733bcfg2+4707c2fg2+15100bdfg2+3663cdfg2\
    -13292d2fg2-3879befg2+4495cefg2+10127defg2-3728e2fg2+4641bf2g2+4464cf2g2+\
    9906df2g2+9831ef2g2+8499f3g2+11570bcg3-4681c2g3+10649bdg3+7246cdg3+12129d\
    2g3+12153beg3-7616ceg3+9637deg3-7784e2g3+10969bfg3-13584cfg3-15482dfg3-11\
    309efg3+7318f2g3+14448bg4-6781cg4-841dg4+1711eg4-2832fg4-7622g5
 ↦ j[29]=d3e2+14778cde2f-3114d2e2f-13169be3f-13347ce3f+4016de3f-8773e4f+3949\
    c3f2+14217bcdf2-6206c2df2+15851bd2f2-4631cd2f2+4035d3f2+2400bcef2-13334c2\
    ef2-15857bdef2-2721cdef2+15432d2ef2+438be2f2-8727ce2f2+372de2f2-9241e3f2+\
    10726bcf3+6c2f3-2177bdf3-14019cdf3+3596d2f3+14931bef3+6183cef3-9197def3+5\
    15e2f3-4159bf4+2375cf4+12338df4-2021ef4-12547f5-10066c4g+2818bd3g-9586cd3\
    g+9397d4g+12851c3eg-14bcdeg-14502c2deg-14125bd2eg-13605cd2eg-1436d3eg-504\
    3bce2g+5367c2e2g+4829bde2g-8515cde2g-6616d2e2g+3543be3g+15428ce3g+12529de\
    3g-10738e4g+1364c3fg-14856bcdfg+9656c2dfg+12265bd2fg+14942cd2fg+242d3fg+6\
    844bcefg+1678c2efg+2318bdefg-14151cdefg-3989d2efg+11254be2fg+15545ce2fg-1\
    369de2fg+6533e3fg-13098bcf2g-15590c2f2g+4805bdf2g+4725cdf2g-2745d2f2g-634\
    5bef2g-6249cef2g+2189def2g-670e2f2g-6085bf3g+11618cf3g+13756df3g-11641ef3\
    g-3341f4g+15536c3g2-11424bcdg2+7915c2dg2-9429bd2g2+3465cd2g2-604d3g2+2161\
    bceg2+8410c2eg2+12016bdeg2-14940cdeg2-1504d2eg2-2763be2g2-8038ce2g2-12724\
    de2g2-11993e3g2-13583bcfg2+12637c2fg2+14155bdfg2-12104cdfg2-6540d2fg2-157\
    23befg2+11076cefg2+13356defg2-5484e2fg2-7366bf2g2+10350cf2g2-11428df2g2-9\
    9ef2g2+2059f3g2+8761bcg3+7144c2g3-9208bdg3+9497cdg3+4458d2g3+12368beg3+13\
    142ceg3-6433deg3+7461e2g3+4324bfg3-5248cfg3-4584dfg3-4610efg3-474f2g3+340\
    0bg4-14639cg4-14762dg4+5613eg4-13327fg4-9319g5
 ↦ j[30]=cd2e2-8628cde2f+942d2e2f+565be3f+15415ce3f-596de3f+12039e4f+699c3f2\
    +14967bcdf2-6413c2df2-5184bd2f2-6556cd2f2-14120d3f2+13736bcef2+11612c2ef2\
    +5344bdef2+11525cdef2-15955d2ef2-14671be2f2+13798ce2f2-10893de2f2-7294e3f\
    2+2995bcf3+14287c2f3+11833bdf3+2664cdf3-11696d2f3-1349bef3-3745cef3-2814d\
    ef3+12591e2f3+6018bf4-464cf4+10512df4+5105ef4-4481f5-4279c4g-11462bd3g-68\
    00cd3g-15912d4g+15405c3eg-7737bcdeg+8935c2deg+11857bd2eg+934cd2eg-1797d3e\
    g-13794bce2g-15117c2e2g+4184bde2g-9521cde2g+15422d2e2g+127be3g+1873ce3g-1\
    0822de3g-5805e4g+14775c3fg-1644bcdfg-5936c2dfg+2909bd2fg+13557cd2fg-3341d\
    3fg-14137bcefg+10143c2efg+14827bdefg-12071cdefg-8287d2efg+7316be2fg+7535c\
    e2fg+4455de2fg-12631e3fg+13815bcf2g+13960c2f2g+4175bdf2g+9936cdf2g+11846d\
    2f2g-9021bef2g-422cef2g+2520def2g-2638e2f2g+8519bf3g+6906cf3g-14635df3g-8\
    216ef3g+8303f4g+14130c3g2-7903bcdg2+3982c2dg2+14679bd2g2+9648cd2g2-2610d3\
    g2-1848bceg2-7366c2eg2-7145bdeg2+3344cdeg2-7653d2eg2+9789be2g2-7797ce2g2-\
    14391de2g2-4406e3g2+14817bcfg2+15476c2fg2+12390bdfg2-15589cdfg2-14063d2fg\
    2+2328befg2-11411cefg2-45defg2+5764e2fg2-11570bf2g2-5597cf2g2-6034df2g2-1\
    72ef2g2+711f3g2-1875bcg3+9424c2g3-883bdg3+13554cdg3-5666d2g3-9578beg3+158\
    32ceg3-731deg3-12300e2g3-3418bfg3+3176cfg3+729dfg3+13379efg3+11547f2g3+10\
    035bg4-15729cg4+14066dg4+7157eg4+12149fg4+15688g5
 ↦ j[31]=bd2e2+10085cde2f-6317d2e2f+8771be3f-453ce3f+12193de3f-6255e4f-5015c\
    3f2+11776bcdf2-14718c2df2+2526bd2f2-4375cd2f2-6631d3f2-660bcef2-6891c2ef2\
    -15412bdef2-11827cdef2+11899d2ef2-9150be2f2-8737ce2f2+10222de2f2+11261e3f\
    2-11290bcf3-6449c2f3-3864bdf3+9375cdf3+14603d2f3+7420bef3-553cef3+171def3\
    +7934e2f3-3486bf4+5965cf4-11672df4-2452ef4-8080f5-11244c4g-12230bd3g+1176\
    3cd3g+2843d4g-213c3eg+9276bcdeg+6703c2deg+13987bd2eg-15509cd2eg-10474d3eg\
    +4537bce2g-7047c2e2g-3276bde2g-10812cde2g-7487d2e2g-13968be3g-8137ce3g+69\
    79de3g-12940e4g-11605c3fg+12175bcdfg+12602c2dfg+6246bd2fg-6215cd2fg-8248d\
    3fg+15710bcefg-5568c2efg-3630bdefg-6193cdefg-5074d2efg+7671be2fg-1526ce2f\
    g+11799de2fg+1216e3fg-1934bcf2g-13671c2f2g-7311bdf2g-7111cdf2g-1671d2f2g-\
    11410bef2g-1632cef2g+11958def2g-5025e2f2g+5189bf3g+15544cf3g-2865df3g-318\
    ef3g-15914f4g-9313c3g2-10827bcdg2+7917c2dg2-7933bd2g2+409cd2g2-4279d3g2+1\
    000bceg2-1708c2eg2-12567bdeg2-9301cdeg2+1356d2eg2-15198be2g2-10659ce2g2+7\
    325de2g2-2041e3g2-11464bcfg2-382c2fg2-3533bdfg2+5606cdfg2+5020d2fg2-6702b\
    efg2+15601cefg2+1494defg2-9024e2fg2+15701bf2g2-11933cf2g2-4204df2g2+38ef2\
    g2-1148f3g2+2550bcg3+10342c2g3-15142bdg3+10134cdg3+3866d2g3-2540beg3+1343\
    6ceg3+7887deg3-12093e2g3+12815bfg3-4754cfg3-12861dfg3-4295efg3-15150f2g3+\
    14173bg4+9126cg4+9996dg4-3858eg4-2944fg4-15809g5
 ↦ j[32]=c2de2-433cde2f-7375d2e2f-207be3f+9390ce3f+1722de3f+9961e4f-7916c3f2\
    -6348bcdf2-467c2df2-2252bd2f2+422cd2f2+9458d3f2-986bcef2+9105c2ef2+5890bd\
    ef2-14451cdef2+13469d2ef2+7771be2f2-3269ce2f2-7137de2f2-1921e3f2-13271bcf\
    3+12838c2f3+6091bdf3+12998cdf3-7322d2f3+3659bef3-13821cef3-5191def3+14637\
    e2f3-10458bf4-7438cf4+14260df4+11ef4+5240f5-14927c4g+7275bd3g+11373cd3g+2\
    044d4g-14840c3eg+654bcdeg+14259c2deg+3245bd2eg+4226cd2eg-8083d3eg+13657bc\
    e2g+1751c2e2g+12575bde2g-8130cde2g-10994d2e2g+10879be3g-948ce3g-15454de3g\
    +11416e4g+13053c3fg+11298bcdfg-11773c2dfg-9885bd2fg-7969cd2fg-2350d3fg+84\
    12bcefg-15478c2efg+3345bdefg+14195cdefg-1826d2efg+3799be2fg-9119ce2fg+110\
    80de2fg-501e3fg-3498bcf2g-3845c2f2g+12628bdf2g-5178cdf2g-10764d2f2g+14553\
    bef2g-2756cef2g+3120def2g-15599e2f2g-4874bf3g+14636cf3g-8005df3g+14696ef3\
    g+11124f4g-6430c3g2+9587bcdg2-13698c2dg2+9007bd2g2+3111cd2g2+13196d3g2-10\
    607bceg2+11739c2eg2-426bdeg2+998cdeg2-9466d2eg2-14321be2g2-6926ce2g2-3278\
    de2g2+15187e3g2-3289bcfg2+14816c2fg2+9844bdfg2-9543cdfg2+12409d2fg2+15217\
    befg2-7851cefg2+7215defg2+15881e2fg2-2863bf2g2-10912cf2g2-7112df2g2+9428e\
    f2g2-1488f3g2-8751bcg3-12403c2g3+8237bdg3+13176cdg3+10991d2g3+14034beg3+4\
    313ceg3+10476deg3-825e2g3-2375bfg3+14049cfg3+6852dfg3-6374efg3-3108f2g3-1\
    237bg4-6069cg4-12334dg4-15952eg4-6672fg4+13402g5
 ↦ j[33]=bcde2+2077cde2f+2789d2e2f+6285be3f+13607ce3f+6204de3f+11616e4f-829c\
    3f2+3834bcdf2-14613c2df2-702bd2f2+6848cd2f2+2569d3f2+4965bcef2-7492c2ef2+\
    543bdef2-12629cdef2-3320d2ef2-4498be2f2+14651ce2f2-994de2f2-4287e3f2+1406\
    7bcf3+13800c2f3-14148bdf3-10698cdf3-6795d2f3+10258bef3+15744cef3-13294def\
    3-6108e2f3+7421bf4-10860cf4+10913df4+7320ef4+12109f5-10226c4g-9873bd3g-99\
    46cd3g+1439d4g+3248c3eg+9622bcdeg+6067c2deg-12232bd2eg-2566cd2eg+10419d3e\
    g-14322bce2g-15434c2e2g-7883bde2g-14823cde2g-9906d2e2g+13298be3g-15723ce3\
    g-15727de3g+11496e4g+13460c3fg+178bcdfg+13862c2dfg-12872bd2fg+11085cd2fg-\
    13541d3fg+9380bcefg+5400c2efg+6724bdefg+15501cdefg-13894d2efg-12008be2fg+\
    11226ce2fg+10049de2fg+9041e3fg+3313bcf2g-2432c2f2g+15447bdf2g+15668cdf2g+\
    7598d2f2g+9979bef2g-13756cef2g-2249def2g-15925e2f2g-11611bf3g-11444cf3g+1\
    5626df3g-42ef3g+11298f4g-4098c3g2+5127bcdg2-9895c2dg2-15439bd2g2+5834cd2g\
    2+3846d3g2-2915bceg2-4955c2eg2-12709bdeg2-7996cdeg2+8882d2eg2-6854be2g2-1\
    3400ce2g2-13692de2g2-14267e3g2+10813bcfg2-15102c2fg2-10117bdfg2-12968cdfg\
    2+13250d2fg2+2770befg2-6371cefg2-7125defg2-13320e2fg2-14436bf2g2+8922cf2g\
    2-15223df2g2+6439ef2g2-3659f3g2+12576bcg3-12956c2g3-405bdg3+15314cdg3+152\
    09d2g3-3850beg3-13880ceg3-6296deg3+8898e2g3-738bfg3-7522cfg3+6570dfg3-991\
    1efg3-4256f2g3-10992bg4-9859cg4-11849dg4+3884eg4-11264fg4+3630g5
 ↦ j[34]=c3e2-11434cde2f+2726d2e2f-4225be3f+10915ce3f-10910de3f+8932e4f-3967\
    c3f2-7499bcdf2+8859c2df2-15841bd2f2-12772cd2f2-8050d3f2-12686bcef2-14015c\
    2ef2+14237bdef2+6544cdef2-2699d2ef2-2553be2f2+12449ce2f2-9380de2f2+7842e3\
    f2-7579bcf3-6609c2f3+15670bdf3-10846cdf3+4612d2f3-770bef3-4981cef3+14346d\
    ef3+5631e2f3-12330bf4-6232cf4-1890df4-2307ef4-14189f5+12132c4g-13537bd3g+\
    8771cd3g-7574d4g-12604c3eg-13765bcdeg-1074c2deg-12173bd2eg+8432cd2eg-7823\
    d3eg-14163bce2g+9709c2e2g+4551bde2g+11626cde2g-5418d2e2g-8758be3g+10225ce\
    3g+8641de3g-350e4g+8569c3fg-2574bcdfg+1338c2dfg+10200bd2fg+15370cd2fg-17d\
    3fg-7601bcefg+6705c2efg-7042bdefg+5577cdefg+10768d2efg+15238be2fg+13388ce\
    2fg-1702de2fg-12146e3fg-5655bcf2g+8900c2f2g-2117bdf2g-8645cdf2g-15222d2f2\
    g+14055bef2g-1781cef2g-7263def2g-1364e2f2g-5313bf3g+3044cf3g-8604df3g+158\
    5ef3g-11582f4g+7431c3g2-12367bcdg2-4951c2dg2+9428bd2g2+1680cd2g2+12814d3g\
    2+13965bceg2-15517c2eg2-4933bdeg2-5358cdeg2+12312d2eg2-10533be2g2-11948ce\
    2g2+3123de2g2-11187e3g2-1012bcfg2+5728c2fg2-3398bdfg2-805cdfg2+7037d2fg2+\
    15897befg2+11738cefg2-9443defg2+2794e2fg2+6118bf2g2-7945cf2g2+10132df2g2-\
    4715ef2g2-766f3g2-5087bcg3+14c2g3-3239bdg3-5831cdg3-4215d2g3+9428beg3+136\
    09ceg3-15754deg3-10834e2g3-238bfg3+903cfg3-8867dfg3-289efg3-15855f2g3+419\
    5bg4-5510cg4+1225dg4-15504eg4-3819fg4+380g5
 ↦ j[35]=d4e+8045cde2f+6931d2e2f+4847be3f+10826ce3f+13320de3f-4586e4f+13524c\
    3f2-14186bcdf2-7492c2df2+7605bd2f2-5372cd2f2+677d3f2-8798bcef2+1519c2ef2+\
    11377bdef2-10743cdef2-8204d2ef2-2032be2f2-9606ce2f2-1722de2f2-11558e3f2+1\
    5330bcf3-6383c2f3+1323bdf3-1616cdf3+6022d2f3+7581bef3+10611cef3-8171def3+\
    5943e2f3+493bf4+10246cf4-13718df4-5602ef4-6434f5+1007c4g+732bd3g-898cd3g-\
    12935d4g-11619c3eg-11984bcdeg-15074c2deg-14890bd2eg+5908cd2eg+8075d3eg+38\
    04bce2g-10819c2e2g+3687bde2g+14669cde2g-3234d2e2g-9745be3g-7758ce3g-981de\
    3g+389e4g-1918c3fg+4383bcdfg+10047c2dfg-5103bd2fg+9341cd2fg-14370d3fg+598\
    4bcefg-1890c2efg+7792bdefg+11684cdefg-14510d2efg+15933be2fg+5012ce2fg+741\
    6de2fg-13231e3fg+2459bcf2g+6903c2f2g-5105bdf2g+5301cdf2g-14769d2f2g+15493\
    bef2g-15328cef2g+3441def2g-9776e2f2g+10234bf3g+9476cf3g-11720df3g+2409ef3\
    g+11290f4g-2554c3g2+11813bcdg2+3787c2dg2-9913bd2g2+11743cd2g2+10811d3g2-5\
    858bceg2-9974c2eg2+12799bdeg2-12588cdeg2-13894d2eg2+5209be2g2-6912ce2g2+1\
    1171de2g2-11593e3g2+11203bcfg2-10106c2fg2-4152bdfg2+1621cdfg2+6972d2fg2-1\
    846befg2+955cefg2+1734defg2-5827e2fg2+452bf2g2-13047cf2g2+13055df2g2+1340\
    5ef2g2-7606f3g2+5624bcg3-3155c2g3-1939bdg3+9838cdg3-14929d2g3+14178beg3+4\
    471ceg3-5172deg3+13976e2g3-2269bfg3+10707cfg3-9861dfg3-6666efg3-5928f2g3+\
    13494bg4+15880cg4+14976dg4-11010eg4+15663fg4-11945g5
 ↦ j[36]=cd3e+4164cde2f-4930d2e2f+3210be3f+3237ce3f+10270de3f-1747e4f-12028c\
    3f2-10171bcdf2-4117c2df2-3246bd2f2+10118cd2f2+13368d3f2+15093bcef2+2511c2\
    ef2-1532bdef2-5068cdef2+4553d2ef2-9879be2f2-7174ce2f2+7979de2f2+5294e3f2-\
    63bcf3-13802c2f3-5402bdf3-7428cdf3-15241d2f3+8847bef3-13953cef3-15523def3\
    +13572e2f3+15029bf4-2661cf4-1177df4+5739ef4+4410f5+8778c4g-815bd3g+13969c\
    d3g+3249d4g-4983c3eg-1679bcdeg-13644c2deg-15272bd2eg+12666cd2eg+681d3eg+1\
    1899bce2g+7945c2e2g+4618bde2g-13102cde2g-5212d2e2g+1050be3g+5569ce3g-1061\
    6de3g-750e4g+2998c3fg-12846bcdfg+8679c2dfg-10221bd2fg-5335cd2fg+15700d3fg\
    -13960bcefg+5095c2efg+7356bdefg+1978cdefg+10838d2efg+12132be2fg-2823ce2fg\
    -5927de2fg-3407e3fg-8306bcf2g+7339c2f2g-4316bdf2g-7982cdf2g-14844d2f2g-11\
    321bef2g+5074cef2g+9430def2g-13286e2f2g-215bf3g+2453cf3g-971df3g+10700ef3\
    g+2676f4g+3403c3g2-6881bcdg2-1291c2dg2+13601bd2g2+2934cd2g2+2275d3g2+9122\
    bceg2-1258c2eg2+9061bdeg2-10314cdeg2-1120d2eg2-6429be2g2-9267ce2g2-12371d\
    e2g2+14370e3g2-11748bcfg2+7707c2fg2+9400bdfg2+12750cdfg2-12760d2fg2+4074b\
    efg2-13336cefg2+14920defg2-14595e2fg2+10989bf2g2-14299cf2g2+11233df2g2+92\
    39ef2g2-2551f3g2-8309bcg3-4989c2g3-1347bdg3-13797cdg3-6584d2g3+11848beg3-\
    1719ceg3+2575deg3-13423e2g3+6996bfg3+141cfg3+3201dfg3+6424efg3+9715f2g3-1\
    0367bg4+11912cg4-13117dg4-14989eg4-2553fg4+6229g5
 ↦ j[37]=bd3e+15585cde2f+6620d2e2f-7236be3f-1779ce3f-3872de3f-11984e4f+6234c\
    3f2-9765bcdf2-13477c2df2-7706bd2f2-10200cd2f2-1451d3f2+4482bcef2+13094c2e\
    f2-10213bdef2+5773cdef2+13283d2ef2+2117be2f2+5240ce2f2+6017de2f2-7063e3f2\
    -8750bcf3-2512c2f3+7988bdf3-4476cdf3+6604d2f3-6655bef3-6050cef3-1868def3-\
    12212e2f3-14623bf4-6776cf4-8050df4-15500ef4+2899f5-13239c4g-8475bd3g-1339\
    cd3g-4239d4g+12117c3eg-14011bcdeg+10172c2deg+373bd2eg-5140cd2eg+9034d3eg+\
    14434bce2g+3667c2e2g-2452bde2g-5977cde2g+13677d2e2g-2029be3g+15030ce3g+11\
    827de3g+9382e4g+13849c3fg+8030bcdfg-8169c2dfg-12944bd2fg+10314cd2fg-7734d\
    3fg+9495bcefg-12263c2efg-6997bdefg+506cdefg+9177d2efg+2084be2fg-14985ce2f\
    g+6342de2fg+2308e3fg+12274bcf2g-8819c2f2g-2432bdf2g-2979cdf2g-13364d2f2g+\
    12853bef2g-11379cef2g-9741def2g-8147e2f2g-3723bf3g-4564cf3g-4420df3g-1564\
    4ef3g+6496f4g-1158c3g2+12976bcdg2-7856c2dg2-7785bd2g2-9554cd2g2+5656d3g2+\
    8622bceg2+15895c2eg2-4667bdeg2-6685cdeg2-1292d2eg2-15998be2g2+13830ce2g2+\
    14858de2g2+7215e3g2+10226bcfg2+4896c2fg2-15510bdfg2-8101cdfg2+290d2fg2-14\
    050befg2-13100cefg2+11416defg2-12178e2fg2-9546bf2g2+6626cf2g2-7988df2g2-6\
    908ef2g2+10113f3g2+3104bcg3-2605c2g3-2716bdg3-10512cdg3+13282d2g3+5983beg\
    3+2883ceg3-15920deg3-9615e2g3-10170bfg3+13107cfg3+15511dfg3+6932efg3+4959\
    f2g3-12598bg4+12426cg4-12774dg4+4956eg4+11380fg4-5820g5
 ↦ j[38]=c4e-2049cde2f-7326d2e2f-10573be3f+2613ce3f-11325de3f+14057e4f-7555c\
    3f2+11545bcdf2-6424c2df2+4808bd2f2-4641cd2f2-14450d3f2-485bcef2-13326c2ef\
    2-3488bdef2+10652cdef2+2864d2ef2-10458be2f2+7572ce2f2-15303de2f2+13349e3f\
    2-10148bcf3+3156c2f3+6222bdf3+14603cdf3-10387d2f3-8676bef3-10860cef3+3787\
    def3+8780e2f3+5532bf4+13791cf4+8603df4-7659ef4-13911f5-11994c4g+9218bd3g+\
    10664cd3g+3066d4g-833c3eg-3708bcdeg+9807c2deg+5667bd2eg-157cd2eg-6624d3eg\
    -10571bce2g-13050c2e2g-3278bde2g-1891cde2g+10956d2e2g+12443be3g+4867ce3g+\
    14583de3g+15518e4g-12362c3fg-3717bcdfg-763c2dfg-2184bd2fg-11926cd2fg+1251\
    3d3fg+14724bcefg+15623c2efg+14423bdefg+3641cdefg+11312d2efg-13701be2fg-15\
    554ce2fg+11256de2fg-3492e3fg-3682bcf2g-1533c2f2g+3204bdf2g+6083cdf2g+1323\
    8d2f2g-7588bef2g+10616cef2g+4352def2g+6504e2f2g+1000bf3g-14687cf3g-2383df\
    3g+12925ef3g+6887f4g+2042c3g2+14586bcdg2-8333c2dg2-14248bd2g2-13584cd2g2+\
    4379d3g2-2184bceg2+12643c2eg2-9033bdeg2-10390cdeg2+12413d2eg2-3324be2g2+9\
    355ce2g2+14851de2g2+10590e3g2-3360bcfg2+1060c2fg2-14774bdfg2-10875cdfg2+1\
    3180d2fg2-12207befg2-9336cefg2+5625defg2-10581e2fg2-14232bf2g2+7978cf2g2+\
    5621df2g2+8645ef2g2-1214f3g2+12086bcg3-10764c2g3+11197bdg3-1416cdg3-7228d\
    2g3+14306beg3+8029ceg3+147deg3+12206e2g3+13160bfg3-8463cfg3-4398dfg3+6510\
    efg3+14733f2g3+11498bg4-7351cg4+15056dg4-15654eg4+14335fg4+2427g5
 ↦ j[39]=d5-15168cde2f+3159d2e2f+1949be3f+14043ce3f-3040de3f+9184e4f+2047c3f\
    2-11177bcdf2+3537c2df2-3258bd2f2+7956cd2f2+12260d3f2+10841bcef2+4185c2ef2\
    -6750bdef2+1578cdef2+12262d2ef2-15506be2f2-13776ce2f2-11480de2f2+10151e3f\
    2+7725bcf3-7251c2f3+1408bdf3-4974cdf3-4406d2f3-5569bef3-3175cef3+15239def\
    3-1082e2f3+5405bf4+1808cf4+12728df4+13678ef4-5938f5+1694c4g-6857bd3g-1083\
    4cd3g-5318d4g+13531c3eg-9105bcdeg+6836c2deg-1784bd2eg-279cd2eg-15693d3eg+\
    12272bce2g+5390c2e2g-2183bde2g+13801cde2g-11427d2e2g+12799be3g+9491ce3g+1\
    4226de3g-15113e4g-12536c3fg-12859bcdfg-2109c2dfg+14057bd2fg-2063cd2fg+138\
    30d3fg-8349bcefg+17c2efg-15169bdefg-13211cdefg-9587d2efg-15565be2fg-1280c\
    e2fg+9425de2fg+1968e3fg-7558bcf2g-12784c2f2g+5414bdf2g+7901cdf2g+9260d2f2\
    g-6639bef2g-11466cef2g+13188def2g+7942e2f2g+12889bf3g-13145cf3g-4525df3g+\
    2069ef3g+11088f4g-1430c3g2-6215bcdg2-1256c2dg2-12247bd2g2+1044cd2g2-3865d\
    3g2-4627bceg2+7041c2eg2+10011bdeg2+9054cdeg2+12731d2eg2+255be2g2-15539ce2\
    g2-12119de2g2-7266e3g2+14218bcfg2+636c2fg2+4028bdfg2-2160cdfg2-5281d2fg2+\
    6543befg2-11864cefg2-12915defg2-8117e2fg2-7589bf2g2+4191cf2g2-10618df2g2-\
    2457ef2g2+7720f3g2+11173bcg3-3332c2g3+15451bdg3-5048cdg3-2631d2g3+12911be\
    g3-13774ceg3-8993deg3-2919e2g3-8918bfg3-2442cfg3+6950dfg3+10639efg3-387f2\
    g3+8575bg4+9643cg4-11324dg4+9266eg4-13885fg4-5109g5
 ↦ j[40]=cd4+5627cde2f-4828d2e2f-10130be3f-13957ce3f+6572de3f+13887e4f+3012c\
    3f2+5585bcdf2-5375c2df2-15476bd2f2-727cd2f2-10898d3f2+3400bcef2-15405c2ef\
    2+9947bdef2+14208cdef2+1621d2ef2-13560be2f2-8513ce2f2-7894de2f2-7978e3f2-\
    2171bcf3+8552c2f3+13135bdf3+10454cdf3-3060d2f3-3452bef3-1638cef3-13088def\
    3-13179e2f3-13316bf4-15123cf4+12463df4+9796ef4+15892f5-10666c4g+8810bd3g+\
    2189cd3g+12857d4g-4929c3eg+9066bcdeg-970c2deg-14960bd2eg-6795cd2eg-7567d3\
    eg-15819bce2g-9300c2e2g+4402bde2g+13891cde2g-2966d2e2g+1632be3g+3687ce3g-\
    11696de3g+14667e4g-10988c3fg-14422bcdfg+5287c2dfg+5403bd2fg-2489cd2fg-136\
    01d3fg-14256bcefg-1462c2efg-10351bdefg-2286cdefg+1107d2efg-1879be2fg+5683\
    ce2fg+15487de2fg+8532e3fg-1039bcf2g-6176c2f2g+12814bdf2g+12643cdf2g-14964\
    d2f2g+15409bef2g-15312cef2g-3318def2g+5270e2f2g-607bf3g+8476cf3g+3501df3g\
    +5288ef3g+14664f4g+9738c3g2+2746bcdg2+2376c2dg2-615bd2g2-11568cd2g2-282d3\
    g2-15934bceg2-525c2eg2+7185bdeg2+13482cdeg2-15291d2eg2+9193be2g2+15237ce2\
    g2-15673de2g2-14105e3g2+807bcfg2+9511c2fg2+12547bdfg2-4284cdfg2+9853d2fg2\
    +14570befg2+12576cefg2+13505defg2+3568e2fg2+2773bf2g2+5472cf2g2-1148df2g2\
    +8476ef2g2-6791f3g2-86bcg3+12001c2g3-4348bdg3-9404cdg3-1830d2g3-15274beg3\
    +11575ceg3+5812deg3+4833e2g3+7234bfg3-15669cfg3-7850dfg3-13148efg3-4347f2\
    g3-10966bg4+8783cg4-4238dg4-8475eg4-12970fg4+12634g5
 ↦ j[41]=bd4-5475cde2f+6113d2e2f+1236be3f+11914ce3f+14708de3f-390e4f-6751c3f\
    2-12882bcdf2-8073c2df2+5960bd2f2-2583cd2f2-6400d3f2-2667bcef2+10942c2ef2-\
    7132bdef2-5133cdef2-6185d2ef2-12971be2f2-7836ce2f2+5141de2f2+11815e3f2+86\
    17bcf3+10105c2f3+7388bdf3+1692cdf3+9153d2f3-10297bef3+7518cef3+682def3-88\
    37e2f3-13183bf4+400cf4+1059df4-7382ef4+1582f5+14048c4g-13952bd3g+7861cd3g\
    -9678d4g+14339c3eg-1827bcdeg-1543c2deg+9656bd2eg+10919cd2eg-8391d3eg-1049\
    1bce2g+2878c2e2g-737bde2g+14792cde2g+6129d2e2g+2302be3g-422ce3g-8013de3g-\
    10045e4g+1990c3fg+11443bcdfg+8186c2dfg+3947bd2fg+10840cd2fg-12378d3fg-288\
    9bcefg-195c2efg+7008bdefg+11938cdefg+7668d2efg-15014be2fg-5739ce2fg+732de\
    2fg-11255e3fg+15756bcf2g-12627c2f2g-11824bdf2g-13006cdf2g-2022d2f2g-4661b\
    ef2g+10279cef2g-5184def2g-13461e2f2g+7428bf3g+3522cf3g-7828df3g-9630ef3g+\
    5254f4g-10722c3g2+5830bcdg2-15001c2dg2+2871bd2g2-1333cd2g2+5942d3g2-4191b\
    ceg2+15878c2eg2-2863bdeg2+9861cdeg2+14677d2eg2-6672be2g2-2236ce2g2-12503d\
    e2g2-4363e3g2-1319bcfg2-15923c2fg2+1327bdfg2-10617cdfg2-5406d2fg2+9986bef\
    g2+13860cefg2+9379defg2+2961e2fg2-15377bf2g2-1982cf2g2+4517df2g2+11642ef2\
    g2+13303f3g2-8255bcg3+4989c2g3+7646bdg3+3092cdg3-15168d2g3-13798beg3-6151\
    ceg3-8667deg3-13687e2g3+817bfg3-3879cfg3+302dfg3+15699efg3-5184f2g3-223bg\
    4-8315cg4+2750dg4-11635eg4-11711fg4+7894g5
 ↦ j[42]=c4d-6838cde2f-12575d2e2f-4314be3f+5389ce3f-4665de3f-10315e4f+10370c\
    3f2+8148bcdf2+1022c2df2+12956bd2f2-7418cd2f2-9669d3f2-2293bcef2-4547c2ef2\
    -12684bdef2-5889cdef2-5650d2ef2+14762be2f2+15112ce2f2-6069de2f2+12381e3f2\
    +13475bcf3-7427c2f3-453bdf3+8550cdf3-7850d2f3+15106bef3+6862cef3-7071def3\
    -1213e2f3-15481bf4+9436cf4+15922df4-3488ef4+15312f5-1519c4g-9492bd3g+1263\
    3cd3g+8777d4g-3040c3eg-10513bcdeg+4423c2deg+10305bd2eg-13846cd2eg+108d3eg\
    +10171bce2g+2827c2e2g-14304bde2g+10615cde2g-13508d2e2g+14030be3g+13666ce3\
    g-12660de3g+3020e4g-13044c3fg-651bcdfg+9597c2dfg-15046bd2fg-6696cd2fg-584\
    9d3fg-13860bcefg+3231c2efg+12792bdefg+2689cdefg-6474d2efg+2660be2fg-6027c\
    e2fg-7237de2fg+15344e3fg-6913bcf2g-12174c2f2g-13502bdf2g-8970cdf2g-11348d\
    2f2g+15466bef2g+14845cef2g-12534def2g-4372e2f2g-3530bf3g-3037cf3g-197df3g\
    +14779ef3g-5128f4g+8744c3g2+4326bcdg2+1648c2dg2-13667bd2g2-10907cd2g2-496\
    1d3g2+10899bceg2-1423c2eg2+7997bdeg2+14569cdeg2+544d2eg2+13933be2g2+3497c\
    e2g2+968de2g2-5393e3g2-12277bcfg2+13807c2fg2-8028bdfg2-14736cdfg2-11437d2\
    fg2+12095befg2-9236cefg2-14795defg2-13347e2fg2-1669bf2g2-1167cf2g2+4446df\
    2g2+7267ef2g2-2294f3g2-4038bcg3-4033c2g3+13727bdg3+9740cdg3-9078d2g3+4882\
    beg3-8041ceg3-13699deg3-12231e2g3+11009bfg3-12366cfg3-2676dfg3+12465efg3+\
    2485f2g3-14021bg4+2583cg4+4004dg4-6733eg4+10413fg4-3203g5
 ↦ j[43]=c5-627cde2f+12983d2e2f+13461be3f-12722ce3f-1902de3f-13705e4f+10635c\
    3f2-4278bcdf2-10929c2df2-7789bd2f2-2142cd2f2-15773d3f2-13716bcef2-13959c2\
    ef2-7597bdef2-947cdef2-7258d2ef2+8759be2f2+5114ce2f2+8895de2f2+11401e3f2+\
    12275bcf3-5825c2f3+1500bdf3-4120cdf3-1790d2f3+15043bef3+10326cef3+6853def\
    3+5874e2f3+10300bf4+7095cf4+2349df4+8876ef4-7970f5+5890c4g-12962bd3g-1260\
    1cd3g-11920d4g-2991c3eg+3603bcdeg+1229c2deg-10323bd2eg+1979cd2eg-12915d3e\
    g+12428bce2g+14906c2e2g+8887bde2g-11877cde2g+6510d2e2g-4055be3g+1583ce3g+\
    992de3g+4149e4g-9846c3fg+14047bcdfg+12208c2dfg-7503bd2fg+9557cd2fg+2731d3\
    fg+12869bcefg+14959c2efg-8327bdefg-1591cdefg-3061d2efg-12602be2fg-240ce2f\
    g+4886de2fg-15585e3fg+10044bcf2g-1000c2f2g+8183bdf2g-3488cdf2g-7070d2f2g-\
    4620bef2g+9954cef2g+2599def2g+11044e2f2g-13159bf3g-2481cf3g+2941df3g+8983\
    ef3g-11246f4g-11940c3g2-15492bcdg2+11460c2dg2+13951bd2g2-986cd2g2+4262d3g\
    2+7032bceg2+6227c2eg2+15152bdeg2+7191cdeg2+6576d2eg2-8711be2g2-11224ce2g2\
    +50de2g2-6328e3g2-4666bcfg2+9901c2fg2-7087bdfg2-10670cdfg2+3461d2fg2-1576\
    1befg2+6042cefg2-6123defg2-7588e2fg2+4727bf2g2+4734cf2g2-11010df2g2-8384e\
    f2g2+10089f3g2+7875bcg3-11414c2g3+5177bdg3-2621cdg3-1235d2g3-14542beg3-63\
    11ceg3-779deg3+15761e2g3-2102bfg3-9023cfg3-4924dfg3+8477efg3-15267f2g3-10\
    021bg4-9011cg4-12981dg4-7663eg4+13207fg4-15883g5
 ↦ j[44]=f6+14502cde2fg-13623d2e2fg-9624be3fg-5286ce3fg-14763de3fg+6291e4fg-\
    2156c3f2g+7742bcdf2g+3835c2df2g+11697bd2f2g+14900cd2f2g+14877d3f2g-6030bc\
    ef2g-2889c2ef2g+6912bdef2g-7337cdef2g+2188d2ef2g+10167be2f2g-2128ce2f2g+1\
    1496de2f2g+4054e3f2g-12426bcf3g-15116c2f3g-15775bdf3g-15708cdf3g+14809d2f\
    3g+657bef3g+14070cef3g-11852def3g+3338e2f3g+2412bf4g+3376cf4g+5031df4g-27\
    50ef4g+12723f5g+11854c4g2+4722bd3g2-11248cd3g2+3287d4g2-13460c3eg2-8149bc\
    deg2+1741c2deg2+7224bd2eg2+13832cd2eg2+6418d3eg2+7875bce2g2+5633c2e2g2+80\
    89bde2g2+7192cde2g2-7400d2e2g2-9123be3g2-11337ce3g2-11811de3g2+10536e4g2+\
    4977c3fg2-6564bcdfg2+7228c2dfg2+11843bd2fg2-15527cd2fg2+8541d3fg2-12195bc\
    efg2+11788c2efg2+3581bdefg2+7888cdefg2-10970d2efg2+4412be2fg2+844ce2fg2-1\
    3566de2fg2-9888e3fg2-7577bcf2g2-12227c2f2g2+581bdf2g2+11656cdf2g2+2596d2f\
    2g2-10645bef2g2+435cef2g2-13392def2g2+3677e2f2g2-7273bf3g2-13915cf3g2-715\
    df3g2-7365ef3g2-5648f4g2-9594c3g3+4161bcdg3+11103c2dg3-13560bd2g3-7963cd2\
    g3-7958d3g3-13945bceg3+767c2eg3+9170bdeg3-6523cdeg3-7240d2eg3+12757be2g3+\
    11157ce2g3-4977de2g3+12100e3g3+10663bcfg3-4364c2fg3-2571bdfg3+854cdfg3+83\
    72d2fg3-4977befg3-1058cefg3+15557defg3+3565e2fg3+308bf2g3-3915cf2g3-13681\
    df2g3-5937ef2g3-4023f3g3-15717bcg4+6363c2g4+2457bdg4-4cdg4+14306d2g4-1597\
    6beg4-10323ceg4-5485deg4-3967e2g4-9442bfg4+8454cfg4+14089dfg4+15980efg4-1\
    0308f2g4-9867bg5-10800cg5+14702dg5-5322eg5+3080fg5-10544g6
 ↦ j[45]=ef5+11572cde2fg-3802d2e2fg-2963be3fg+7487ce3fg+11634de3fg+1481e4fg-\
    11229c3f2g-11880bcdf2g-8162c2df2g-12739bd2f2g+12994cd2f2g+9979d3f2g-6440b\
    cef2g-11176c2ef2g+1907bdef2g-12041cdef2g-12054d2ef2g-1176be2f2g-15532ce2f\
    2g+14160de2f2g+10163e3f2g-976bcf3g+2177c2f3g-12745bdf3g+10880cdf3g+13603d\
    2f3g+2982bef3g+5961cef3g+12675def3g-10338e2f3g-6602bf4g+11021cf4g-11495df\
    4g+1489ef4g-9504f5g-5237c4g2+561bd3g2-11928cd3g2-12694d4g2+7932c3eg2-68bc\
    deg2-3756c2deg2+4967bd2eg2+15216cd2eg2-14748d3eg2-7021bce2g2+14191c2e2g2-\
    8652bde2g2+5980cde2g2-816d2e2g2+4304be3g2+10151ce3g2+2422de3g2-3515e4g2-1\
    1085c3fg2-8958bcdfg2+567c2dfg2-7399bd2fg2+5056cd2fg2-12629d3fg2+6809bcefg\
    2-5282c2efg2+8263bdefg2+10434cdefg2+3258d2efg2+1034be2fg2-12904ce2fg2-882\
    1de2fg2-8582e3fg2-409bcf2g2+1004c2f2g2-12334bdf2g2-878cdf2g2-14387d2f2g2-\
    6806bef2g2-5246cef2g2+3168def2g2+11740e2f2g2+8194bf3g2-10918cf3g2-1070df3\
    g2+9212ef3g2-9652f4g2-8182c3g3+2811bcdg3-9107c2dg3-10455bd2g3+9066cd2g3+3\
    128d3g3-5914bceg3+11040c2eg3-14079bdeg3-13939cdeg3+15203d2eg3+7591be2g3+1\
    4021ce2g3-3242de2g3-3703e3g3-10324bcfg3+13650c2fg3+10080bdfg3+4660cdfg3+4\
    063d2fg3-5284befg3-5497cefg3-9368defg3+9713e2fg3-10972bf2g3-5118cf2g3+494\
    0df2g3+12053ef2g3-6850f3g3+9326bcg4+8297c2g4+6618bdg4+6860cdg4-13338d2g4+\
    5077beg4-14968ceg4-13921deg4+12993e2g4+3449bfg4+1321cfg4+11210dfg4+427efg\
    4+11354f2g4-15786bg5+5829cg5+15966dg5+14358eg5+10407fg5-7447g6
 ↦ j[46]=df5+8372cde2fg-7557d2e2fg-2916be3fg-14748ce3fg+13461de3fg+6868e4fg+\
    8424c3f2g+13049bcdf2g+15488c2df2g+5666bd2f2g+6385cd2f2g+869d3f2g-2522bcef\
    2g+803c2ef2g+1177bdef2g-9076cdef2g+3598d2ef2g+4583be2f2g+3792ce2f2g-14128\
    de2f2g+10031e3f2g+12499bcf3g+13401c2f3g+8767bdf3g+15278cdf3g-9990d2f3g+10\
    137bef3g-10098cef3g-5781def3g+4902e2f3g-14615bf4g+15854cf4g+12939df4g+152\
    47ef4g+5930f5g-65c4g2-1801bd3g2+11942cd3g2-10635d4g2-6238c3eg2-873bcdeg2-\
    9281c2deg2+6800bd2eg2+2304cd2eg2-15896d3eg2-7927bce2g2-13202c2e2g2+11100b\
    de2g2+14583cde2g2+8329d2e2g2-4564be3g2-497ce3g2+12798de3g2+849e4g2-8600c3\
    fg2+8196bcdfg2-3368c2dfg2+14986bd2fg2+1504cd2fg2+9152d3fg2+1433bcefg2-645\
    9c2efg2-1527bdefg2-15663cdefg2-15570d2efg2-1967be2fg2+8328ce2fg2+3274de2f\
    g2+2081e3fg2+3972bcf2g2+740c2f2g2+8364bdf2g2-2361cdf2g2-196d2f2g2-6366bef\
    2g2+1589cef2g2-3750def2g2+7486e2f2g2-6732bf3g2-14923cf3g2+1248df3g2+7699e\
    f3g2-4054f4g2+12374c3g3+4734bcdg3-3992c2dg3-12779bd2g3-384cd2g3+15374d3g3\
    +7746bceg3-10073c2eg3+134bdeg3+9698cdeg3+5786d2eg3+4817be2g3-6167ce2g3-12\
    673de2g3+3806e3g3+12692bcfg3+11727c2fg3+5102bdfg3+12412cdfg3+11381d2fg3+1\
    2100befg3-15719cefg3-4538defg3-7162e2fg3-4984bf2g3-14710cf2g3+5506df2g3-1\
    923ef2g3-479f3g3-9624bcg4+8725c2g4-9934bdg4+7433cdg4+14995d2g4-6971beg4+1\
    3635ceg4-56deg4-824e2g4+7075bfg4+11064cfg4+15125dfg4+2954efg4+5628f2g4-50\
    21bg5+12849cg5+4438dg5+2477eg5-9808fg5+7723g6
 ↦ j[47]=cf5+9474cde2fg+561d2e2fg+2347be3fg+14349ce3fg-6518de3fg-12357e4fg-5\
    741c3f2g+14925bcdf2g-1734c2df2g-15320bd2f2g+14256cd2f2g-11606d3f2g-8670bc\
    ef2g+3016c2ef2g-14893bdef2g+2324cdef2g-13145d2ef2g-9515be2f2g-5606ce2f2g+\
    2145de2f2g-15802e3f2g+10283bcf3g-9873c2f3g+4975bdf3g+7458cdf3g+3238d2f3g-\
    1103bef3g+6049cef3g+9164def3g+400e2f3g-1712bf4g+398cf4g+8465df4g-10618ef4\
    g+9680f5g+8316c4g2+4973bd3g2-2899cd3g2-4130d4g2-14616c3eg2+2275bcdeg2-281\
    c2deg2-10468bd2eg2-2708cd2eg2+10542d3eg2+5362bce2g2-3194c2e2g2+8999bde2g2\
    -7809cde2g2-6406d2e2g2-540be3g2+15623ce3g2-3015de3g2-4305e4g2+916c3fg2-58\
    82bcdfg2+5993c2dfg2+8886bd2fg2-12040cd2fg2-4871d3fg2+12347bcefg2+3812c2ef\
    g2-3705bdefg2+4076cdefg2+6344d2efg2+4269be2fg2-628ce2fg2-1201de2fg2-12450\
    e3fg2+10185bcf2g2+4092c2f2g2-12775bdf2g2-7858cdf2g2+8221d2f2g2+8787bef2g2\
    -6450cef2g2-2696def2g2+13491e2f2g2+1493bf3g2+5885cf3g2-13050df3g2-8641ef3\
    g2-14518f4g2+15558c3g3-2228bcdg3+3724c2dg3+4674bd2g3+5831cd2g3-3351d3g3+2\
    972bceg3-8215c2eg3+9328bdeg3+1950cdeg3+13542d2eg3+1527be2g3-1759ce2g3+814\
    0de2g3-4315e3g3-10513bcfg3+4833c2fg3-1090bdfg3-13876cdfg3-8445d2fg3-10278\
    befg3+15048cefg3+5774defg3+1258e2fg3-4661bf2g3+14491cf2g3+7094df2g3+6021e\
    f2g3+6856f3g3-2977bcg4+4069c2g4-13546bdg4+2713cdg4-12045d2g4+486beg4+1110\
    ceg4-1364deg4-1922e2g4+12135bfg4-6675cfg4-453dfg4+10212efg4-2591f2g4-1352\
    bg5+7842cg5+1407dg5-7611eg5-1673fg5-14701g6
 ↦ j[48]=bf5+14327cde2fg+2563d2e2fg+10420be3fg+14706ce3fg-14235de3fg+14921e4\
    fg+15171c3f2g-5551bcdf2g-14159c2df2g-13104bd2f2g+13569cd2f2g+14064d3f2g-1\
    0178bcef2g-4721c2ef2g+10920bdef2g-7505cdef2g+685d2ef2g+2809be2f2g-7528ce2\
    f2g+14065de2f2g+25e3f2g-8980bcf3g-5216c2f3g-5139bdf3g-1892cdf3g+5835d2f3g\
    +13561bef3g+209cef3g+11471def3g+14425e2f3g-6219bf4g+4351cf4g+10381df4g+56\
    86ef4g+13575f5g+7609c4g2+10536bd3g2-13698cd3g2+777d4g2+6681c3eg2-7613bcde\
    g2-11406c2deg2+3404bd2eg2+10050cd2eg2+10604d3eg2+9077bce2g2+8268c2e2g2-63\
    6bde2g2-2041cde2g2-5080d2e2g2+475be3g2-12261ce3g2-741de3g2-9376e4g2-13068\
    c3fg2+2143bcdfg2+7715c2dfg2+6103bd2fg2+13527cd2fg2+5518d3fg2+8090bcefg2+8\
    235c2efg2-8320bdefg2-13633cdefg2-134d2efg2-2006be2fg2-13934ce2fg2-8011de2\
    fg2+459e3fg2+3388bcf2g2+7635c2f2g2+1990bdf2g2-1607cdf2g2+7231d2f2g2+2053b\
    ef2g2-6278cef2g2-13551def2g2+14684e2f2g2-11980bf3g2+4574cf3g2+4831df3g2+1\
    5812ef3g2-10744f4g2-10797c3g3+12636bcdg3+5313c2dg3+1209bd2g3+2482cd2g3-14\
    162d3g3-4409bceg3-13289c2eg3-13516bdeg3-13000cdeg3-1837d2eg3-3840be2g3-13\
    18ce2g3+9497de2g3-3273e3g3+348bcfg3-10957c2fg3+3561bdfg3+1188cdfg3-5469d2\
    fg3-3278befg3-8783cefg3-10481defg3-8737e2fg3-4365bf2g3+5656cf2g3-695df2g3\
    +10497ef2g3-3192f3g3-7806bcg4+13510c2g4-2086bdg4+12650cdg4-1454d2g4-5489b\
    eg4-13681ceg4+8733deg4-4623e2g4-4985bfg4-5193cfg4+8792dfg4+8864efg4+15030\
    f2g4-5227bg5+15821cg5+13545dg5+9117eg5+6388fg5-1550g6
 ↦ j[49]=e2f4+3118cde2fg-1019d2e2fg+6886be3fg+15789ce3fg+9943de3fg-7500e4fg-\
    6254c3f2g+8075bcdf2g+4581c2df2g+3322bd2f2g+10932cd2f2g-15908d3f2g-15077bc\
    ef2g+5266c2ef2g-12845bdef2g-4385cdef2g-8112d2ef2g-9616be2f2g-11806ce2f2g-\
    13979de2f2g+868e3f2g+961bcf3g-1077c2f3g+15011bdf3g+9433cdf3g+9544d2f3g-73\
    92bef3g+12879cef3g-7872def3g+12698e2f3g-11097bf4g+353cf4g-6967df4g-3895ef\
    4g+8400f5g+10288c4g2+11482bd3g2+3948cd3g2+15560d4g2-8577c3eg2+361bcdeg2+1\
    0350c2deg2-3338bd2eg2-2716cd2eg2+4456d3eg2+1632bce2g2-3153c2e2g2-15036bde\
    2g2-10564cde2g2-9755d2e2g2+781be3g2+14711ce3g2+4798de3g2-10440e4g2+4095c3\
    fg2+3371bcdfg2-3421c2dfg2-9815bd2fg2+2097cd2fg2+1940d3fg2-8040bcefg2-1558\
    8c2efg2-9507bdefg2+6411cdefg2-13433d2efg2+10189be2fg2+3717ce2fg2+15053de2\
    fg2-4519e3fg2+12961bcf2g2+12109c2f2g2-2621bdf2g2-12416cdf2g2-5078d2f2g2-4\
    675bef2g2+6925cef2g2+2313def2g2+10838e2f2g2+11875bf3g2-1196cf3g2+14297df3\
    g2+15363ef3g2+8281f4g2+10562c3g3-10565bcdg3-12655c2dg3+7833bd2g3-7449cd2g\
    3+11218d3g3-5873bceg3-13539c2eg3+1383bdeg3+15296cdeg3+14357d2eg3+7298be2g\
    3-3619ce2g3+3818de2g3+15523e3g3-6646bcfg3-5441c2fg3+11928bdfg3-827cdfg3+2\
    624d2fg3-10089befg3+13653cefg3-502defg3-11095e2fg3-2070bf2g3+4641cf2g3-46\
    33df2g3-10868ef2g3-10999f3g3-12503bcg4+8790c2g4+5632bdg4-7136cdg4-13456d2\
    g4+807beg4-2287ceg4-13448deg4-11630e2g4-6040bfg4+8665cfg4+15998dfg4-7193e\
    fg4-189f2g4+9054bg5-6520cg5-12610dg5-5733eg5+6074fg5+12891g6
 ↦ j[50]=def4-5344cde2fg+7605d2e2fg-877be3fg-13840ce3fg+11126de3fg-3868e4fg+\
    8574c3f2g-7993bcdf2g-13840c2df2g+10268bd2f2g-9112cd2f2g+6660d3f2g-14305bc\
    ef2g-13195c2ef2g+15853bdef2g+765cdef2g-14694d2ef2g-6482be2f2g-13963ce2f2g\
    -894de2f2g+2686e3f2g-10124bcf3g+15907c2f3g-83bdf3g+2303cdf3g-5864d2f3g+60\
    71bef3g-5066cef3g+2164def3g+9838e2f3g+890bf4g-2869cf4g-9800df4g+5477ef4g+\
    8479f5g-2037c4g2-13513bd3g2-2064cd3g2+13981d4g2+3475c3eg2-3038bcdeg2-2307\
    c2deg2+14324bd2eg2+7545cd2eg2+10217d3eg2-624bce2g2-6922c2e2g2-13228bde2g2\
    +6194cde2g2+4163d2e2g2+4930be3g2-5813ce3g2+1622de3g2-11347e4g2-5926c3fg2+\
    7952bcdfg2-8187c2dfg2-750bd2fg2+268cd2fg2-4632d3fg2-4668bcefg2-1101c2efg2\
    +12038bdefg2+10710cdefg2+6781d2efg2-9277be2fg2-15168ce2fg2+15451de2fg2-14\
    495e3fg2-11900bcf2g2-6596c2f2g2+3039bdf2g2-8689cdf2g2-9436d2f2g2-947bef2g\
    2+15383cef2g2-15400def2g2+5548e2f2g2-11498bf3g2-13537cf3g2-12228df3g2+187\
    ef3g2+11055f4g2+10141c3g3+6625bcdg3+5702c2dg3-15308bd2g3+38cd2g3-6162d3g3\
    +12316bceg3+8605c2eg3+302bdeg3-2809cdeg3+1411d2eg3-3058be2g3-9724ce2g3+14\
    162de2g3-1781e3g3+6271bcfg3+12933c2fg3+14189bdfg3-5452cdfg3-13366d2fg3-15\
    587befg3-6785cefg3+2970defg3+3950e2fg3-173bf2g3+3536cf2g3-7023df2g3+11198\
    ef2g3-522f3g3+15488bcg4+14788c2g4+15571bdg4-3292cdg4-8625d2g4-1194beg4-10\
    642ceg4-6995deg4+15839e2g4-9736bfg4+11428cfg4-4365dfg4-7740efg4-37f2g4+82\
    36bg5-3609cg5+10609dg5+2695eg5-1541fg5+2531g6
 ↦ j[51]=cef4+7480cde2fg+10759d2e2fg-2584be3fg-1857ce3fg-13831de3fg-6231e4fg\
    +7713c3f2g+9684bcdf2g-2499c2df2g-13217bd2f2g-1707cd2f2g+12684d3f2g+14388b\
    cef2g+13024c2ef2g+2152bdef2g+13743cdef2g-12086d2ef2g+10093be2f2g-14394ce2\
    f2g+8379de2f2g-6736e3f2g+6659bcf3g+10800c2f3g-15313bdf3g+6518cdf3g+11536d\
    2f3g-2379bef3g+12444cef3g-11602def3g-10980e2f3g-12099bf4g+8734cf4g-10633d\
    f4g+3565ef4g+5515f5g-897c4g2+14702bd3g2+13600cd3g2+12494d4g2-5661c3eg2-64\
    8bcdeg2+5282c2deg2-1015bd2eg2+3932cd2eg2-452d3eg2-7510bce2g2+6083c2e2g2+6\
    723bde2g2+7698cde2g2+14999d2e2g2-5790be3g2-4482ce3g2+5674de3g2-14709e4g2+\
    1869c3fg2-10509bcdfg2+1314c2dfg2+2857bd2fg2-7191cd2fg2-9538d3fg2-13441bce\
    fg2-12249c2efg2+1000bdefg2+12513cdefg2+6740d2efg2+12612be2fg2-13411ce2fg2\
    +1573de2fg2+8873e3fg2+7677bcf2g2-12340c2f2g2-1883bdf2g2+221cdf2g2+12525d2\
    f2g2-3576bef2g2-2664cef2g2-11539def2g2+12577e2f2g2-6509bf3g2+2379cf3g2-13\
    455df3g2-15778ef3g2+10814f4g2-5312c3g3-13267bcdg3-14287c2dg3+1547bd2g3-15\
    126cd2g3-4775d3g3+4273bceg3+130c2eg3-10367bdeg3+1175cdeg3+7553d2eg3-6824b\
    e2g3-2625ce2g3-806de2g3-5468e3g3-12565bcfg3+421c2fg3+10325bdfg3+812cdfg3+\
    15615d2fg3-14827befg3+9202cefg3+5558defg3-6515e2fg3+12866bf2g3-7067cf2g3-\
    7924df2g3-10730ef2g3+2335f3g3+9009bcg4+10447c2g4+11776bdg4-8912cdg4-1575d\
    2g4+1824beg4+1253ceg4+15421deg4+2530e2g4+4223bfg4+4944cfg4+2352dfg4-12440\
    efg4+5625f2g4+9933bg5+3902cg5+8317dg5+8267eg5+6252fg5-1676g6
 ↦ j[52]=bef4+14847cde2fg+11125d2e2fg-9088be3fg+1862ce3fg-12834de3fg-5401e4f\
    g+15361c3f2g+11376bcdf2g+12151c2df2g+4174bd2f2g+6476cd2f2g-9666d3f2g-403b\
    cef2g+13184c2ef2g-1053bdef2g+7191cdef2g+13848d2ef2g+7362be2f2g+10644ce2f2\
    g-5568de2f2g-10138e3f2g-3788bcf3g+1959c2f3g+5404bdf3g+15858cdf3g-11855d2f\
    3g+12775bef3g-9428cef3g-5854def3g-8202e2f3g+11943bf4g-10304cf4g-13901df4g\
    +8310ef4g-2781f5g+4733c4g2+5101bd3g2-782cd3g2-14069d4g2+6632c3eg2+14738bc\
    deg2-507c2deg2+12996bd2eg2+7066cd2eg2-12499d3eg2-3465bce2g2-1342c2e2g2-84\
    36bde2g2+15347cde2g2-10290d2e2g2-4266be3g2-11245ce3g2-8224de3g2-12418e4g2\
    +4142c3fg2-8389bcdfg2-15640c2dfg2-7648bd2fg2-4460cd2fg2-1011d3fg2-10773bc\
    efg2+5900c2efg2-8323bdefg2-14067cdefg2-2004d2efg2+3764be2fg2+11371ce2fg2-\
    4728de2fg2+11302e3fg2-15655bcf2g2+10076c2f2g2+6769bdf2g2-1845cdf2g2-2337d\
    2f2g2-6604bef2g2+4771cef2g2-5106def2g2-3156e2f2g2-327bf3g2-15094cf3g2-107\
    51df3g2+1635ef3g2-1771f4g2+695c3g3-791bcdg3-13822c2dg3-7212bd2g3+627cd2g3\
    +13595d3g3+923bceg3-7628c2eg3+14269bdeg3+8864cdeg3-13804d2eg3-3612be2g3+6\
    491ce2g3-8101de2g3+7807e3g3+10023bcfg3+5321c2fg3+11569bdfg3-7389cdfg3+547\
    3d2fg3-10605befg3+3420cefg3-1577defg3+10438e2fg3-10011bf2g3-15768cf2g3-12\
    469df2g3+11530ef2g3-7019f3g3+958bcg4-2238c2g4+332bdg4+14830cdg4-2433d2g4+\
    14444beg4+12520ceg4+9331deg4-10397e2g4-14869bfg4+5806cfg4+6373dfg4+8681ef\
    g4+8839f2g4-13603bg5+13094cg5-11237dg5+9351eg5+4542fg5-10259g6
 ↦ j[53]=d2f4-9261cde2fg+6594d2e2fg-2412be3fg-13732ce3fg-6560de3fg+12216e4fg\
    -1462c3f2g+11859bcdf2g-6906c2df2g+8428bd2f2g+8904cd2f2g-7071d3f2g-14332bc\
    ef2g+1868c2ef2g+11526bdef2g+10167cdef2g+2454d2ef2g-2697be2f2g+8120ce2f2g+\
    4148de2f2g-12962e3f2g+15235bcf3g-9261c2f3g-1970bdf3g+5428cdf3g+12975d2f3g\
    -15821bef3g+5041cef3g+2728def3g+8417e2f3g-2641bf4g+6278cf4g-10005df4g+414\
    3ef4g-3925f5g-5769c4g2-14390bd3g2-13289cd3g2+3700d4g2+10357c3eg2-10288bcd\
    eg2+351c2deg2-9702bd2eg2-1365cd2eg2+9351d3eg2-14007bce2g2-4891c2e2g2+5021\
    bde2g2-14145cde2g2-3959d2e2g2-6223be3g2+3934ce3g2-13286de3g2+8925e4g2+414\
    5c3fg2-15591bcdfg2-6418c2dfg2+2288bd2fg2-12688cd2fg2+3237d3fg2+5000bcefg2\
    +14560c2efg2+10531bdefg2+13389cdefg2+12625d2efg2-10466be2fg2+7438ce2fg2-1\
    0095de2fg2-7209e3fg2+7240bcf2g2-7899c2f2g2+1294bdf2g2+6766cdf2g2+2697d2f2\
    g2+6535bef2g2+8014cef2g2-6992def2g2+12854e2f2g2+8139bf3g2-3372cf3g2-13639\
    df3g2+550ef3g2+12208f4g2-9661c3g3+1572bcdg3+13842c2dg3-14527bd2g3-9882cd2\
    g3+3173d3g3-15562bceg3-11858c2eg3+4394bdeg3+517cdeg3-5990d2eg3-2513be2g3+\
    9770ce2g3-12022de2g3+6930e3g3+11262bcfg3+6903c2fg3-15795bdfg3-14495cdfg3+\
    4115d2fg3-9387befg3+6754cefg3-13475defg3-12361e2fg3+4627bf2g3+11755cf2g3-\
    14689df2g3+612ef2g3-4635f3g3-12012bcg4+12972c2g4-15718bdg4+3302cdg4+12681\
    d2g4-107beg4+8898ceg4+43deg4+12342e2g4-1247bfg4+12828cfg4+8026dfg4-9496ef\
    g4-1487f2g4-3552bg5-7597cg5-4940dg5+7169eg5+9124fg5-15587g6
 ↦ j[54]=cdf4+15408cde2fg-6670d2e2fg-4370be3fg-7628ce3fg-4890de3fg-8659e4fg-\
    7306c3f2g-14451bcdf2g-14599c2df2g+3857bd2f2g-4946cd2f2g-7638d3f2g-2127bce\
    f2g+11427c2ef2g+15621bdef2g-4801cdef2g+4077d2ef2g+5248be2f2g-13790ce2f2g-\
    14485de2f2g-8046e3f2g+14765bcf3g-6882c2f3g+7006bdf3g-11335cdf3g-5901d2f3g\
    -5930bef3g+15871cef3g+14812def3g+6289e2f3g-14466bf4g+14511cf4g-15201df4g-\
    4042ef4g+14337f5g-7163c4g2-373bd3g2+11265cd3g2-8969d4g2+7649c3eg2+11043bc\
    deg2+13049c2deg2+13263bd2eg2+13449cd2eg2-5201d3eg2+13237bce2g2+5089c2e2g2\
    +344bde2g2-6425cde2g2-2878d2e2g2+4040be3g2+12193ce3g2-10219de3g2-3389e4g2\
    +5582c3fg2+10253bcdfg2-6041c2dfg2+1189bd2fg2+3918cd2fg2-8476d3fg2+3530bce\
    fg2+4103c2efg2+11666bdefg2-3733cdefg2+1043d2efg2+5834be2fg2-15492ce2fg2+1\
    5531de2fg2-11457e3fg2+4513bcf2g2+15933c2f2g2-908bdf2g2+11211cdf2g2-4009d2\
    f2g2-7330bef2g2+6188cef2g2-7044def2g2+11111e2f2g2+1770bf3g2+786cf3g2+4645\
    df3g2-15090ef3g2-12453f4g2-10621c3g3-1691bcdg3-11965c2dg3-1443bd2g3-1329c\
    d2g3-7431d3g3-12429bceg3+7961c2eg3-14299bdeg3-2980cdeg3+14366d2eg3+2611be\
    2g3+15311ce2g3-15606de2g3+103e3g3-2301bcfg3-5944c2fg3-4202bdfg3+4807cdfg3\
    +2477d2fg3-4212befg3+14456cefg3+13885defg3+3666e2fg3+5750bf2g3+7347cf2g3+\
    3920df2g3+14920ef2g3-10694f3g3+15494bcg4-7834c2g4-5018bdg4+1555cdg4-6581d\
    2g4-3903beg4-459ceg4+6107deg4+14870e2g4-15783bfg4+10677cfg4-3400dfg4-1125\
    8efg4+3100f2g4-3795bg5-15353cg5+13820dg5-9064eg5+9218fg5-6667g6
 ↦ j[55]=bdf4+7948cde2fg+14296d2e2fg-11327be3fg-14319ce3fg-10391de3fg-10807e\
    4fg-3037c3f2g+1478bcdf2g-11997c2df2g+8220bd2f2g+11516cd2f2g+9114d3f2g+156\
    23bcef2g+12354c2ef2g-7808bdef2g-6179cdef2g+1349d2ef2g+2871be2f2g-4759ce2f\
    2g+12428de2f2g+10600e3f2g-2132bcf3g+10552c2f3g-132bdf3g-2653cdf3g+8429d2f\
    3g+1776bef3g+5372cef3g+7291def3g-14671e2f3g+204bf4g-6960cf4g-14496df4g-13\
    502ef4g+14857f5g-3023c4g2+7469bd3g2-11940cd3g2+2056d4g2+3756c3eg2-9841bcd\
    eg2+11822c2deg2+7628bd2eg2+13261cd2eg2+12763d3eg2+9862bce2g2+10525c2e2g2-\
    15750bde2g2-7801cde2g2+7160d2e2g2+3275be3g2-6158ce3g2+5194de3g2+3851e4g2+\
    6956c3fg2-12537bcdfg2-679c2dfg2+10196bd2fg2-6572cd2fg2-1553d3fg2-7525bcef\
    g2-3433c2efg2-14264bdefg2+7603cdefg2-869d2efg2+7666be2fg2+11265ce2fg2-145\
    82de2fg2-7227e3fg2-6781bcf2g2+2173c2f2g2-4577bdf2g2-5435cdf2g2+15376d2f2g\
    2-3332bef2g2+1388cef2g2+3222def2g2+9656e2f2g2+14687bf3g2+5236cf3g2-2565df\
    3g2-1056ef3g2-10252f4g2+3477c3g3+12384bcdg3-3904c2dg3-4867bd2g3+5714cd2g3\
    -13012d3g3+3080bceg3+1244c2eg3-4728bdeg3+3673cdeg3+11971d2eg3-11556be2g3+\
    6987ce2g3+13344de2g3+5268e3g3+8496bcfg3-8104c2fg3-12045bdfg3+7053cdfg3+98\
    21d2fg3-12538befg3-2269cefg3+900defg3-10460e2fg3+14948bf2g3+15095cf2g3+12\
    725df2g3-2474ef2g3+13461f3g3-1097bcg4-484c2g4-8528bdg4-13882cdg4+11810d2g\
    4-5311beg4-3843ceg4-12123deg4+3947e2g4-14453bfg4+12304cfg4-7371dfg4+9619e\
    fg4-5075f2g4+9262bg5-3552cg5+561dg5-4213eg5-4109fg5+11460g6
 ↦ j[56]=c2f4-13463cde2fg-7183d2e2fg-2057be3fg-7513ce3fg+2597de3fg+9863e4fg+\
    11184c3f2g-7358bcdf2g-5685c2df2g+7883bd2f2g-12967cd2f2g+9762d3f2g+1551bce\
    f2g-8345c2ef2g-3469bdef2g-13804cdef2g+2910d2ef2g-12217be2f2g-12780ce2f2g+\
    491de2f2g+12898e3f2g+11028bcf3g+2249c2f3g-4831bdf3g+3679cdf3g-1784d2f3g+1\
    442bef3g+6707cef3g+8379def3g-4500e2f3g-11017bf4g+5601cf4g-10187df4g+10728\
    ef4g+7372f5g-5431c4g2-8545bd3g2+3849cd3g2+6509d4g2+11351c3eg2+320bcdeg2+1\
    644c2deg2+10846bd2eg2-3835cd2eg2+14340d3eg2+14664bce2g2-13243c2e2g2+1494b\
    de2g2-5886cde2g2+7252d2e2g2-8006be3g2-844ce3g2-5903de3g2+10449e4g2-15172c\
    3fg2+11132bcdfg2-1554c2dfg2+11056bd2fg2+5142cd2fg2-8136d3fg2+4622bcefg2+9\
    45c2efg2-15604bdefg2-15966cdefg2-9049d2efg2+12702be2fg2+2167ce2fg2-13189d\
    e2fg2+9253e3fg2+1195bcf2g2-7485c2f2g2-4765bdf2g2-14023cdf2g2+7875d2f2g2-1\
    2302bef2g2+6047cef2g2+3425def2g2-9811e2f2g2+2018bf3g2+10574cf3g2+9375df3g\
    2+11409ef3g2+4515f4g2-6099c3g3+8204bcdg3+10598c2dg3+14396bd2g3-7616cd2g3-\
    9002d3g3-11137bceg3-459c2eg3+254bdeg3+12522cdeg3-2673d2eg3-4813be2g3+9572\
    ce2g3+15805de2g3+6425e3g3+10688bcfg3-15428c2fg3+11152bdfg3+14500cdfg3-657\
    9d2fg3+9582befg3-5493cefg3-2125defg3+12301e2fg3+13586bf2g3+15012cf2g3+116\
    98df2g3+7860ef2g3+11018f3g3-10686bcg4-7357c2g4+10875bdg4+7353cdg4-6734d2g\
    4+5103beg4+15351ceg4-10544deg4+15787e2g4-4927bfg4+11938cfg4-3152dfg4+7066\
    efg4-5181f2g4+9947bg5+2544cg5-1557dg5-2289eg5-10571fg5-13289g6
 ↦ j[57]=bcf4-11459cde2fg-2911d2e2fg+11236be3fg+3172ce3fg-12855de3fg-7376e4f\
    g+2281c3f2g+13084bcdf2g+7488c2df2g-8082bd2f2g-14593cd2f2g+2523d3f2g-4933b\
    cef2g+8948c2ef2g+9778bdef2g-14705cdef2g+4617d2ef2g-8309be2f2g-10275ce2f2g\
    +9355de2f2g-413e3f2g+1830bcf3g-1185c2f3g-6755bdf3g-3132cdf3g-12505d2f3g+8\
    350bef3g-14882cef3g+14466def3g+5891e2f3g-2242bf4g+1595cf4g-14309df4g+1228\
    ef4g+3023f5g+2316c4g2-15565bd3g2+9906cd3g2-9128d4g2-760c3eg2-99bcdeg2+706\
    1c2deg2+6851bd2eg2-9568cd2eg2-4315d3eg2-3503bce2g2+3705c2e2g2+15684bde2g2\
    -14606cde2g2+6556d2e2g2-9372be3g2-6595ce3g2-5169de3g2-14139e4g2+7989c3fg2\
    -5291bcdfg2+880c2dfg2-10770bd2fg2-4548cd2fg2+7708d3fg2+6128bcefg2-1892c2e\
    fg2-4692bdefg2+13700cdefg2+13633d2efg2+10168be2fg2+6086ce2fg2+9943de2fg2-\
    14750e3fg2+5120bcf2g2+3104c2f2g2+9890bdf2g2+6084cdf2g2-15289d2f2g2-15431b\
    ef2g2+487cef2g2+12288def2g2-2431e2f2g2-1884bf3g2-2829cf3g2+14534df3g2-136\
    94ef3g2-9976f4g2-9568c3g3-8352bcdg3+8601c2dg3-2891bd2g3-6725cd2g3-2402d3g\
    3-1417bceg3+2972c2eg3+5425bdeg3-12646cdeg3+5138d2eg3+2716be2g3+122ce2g3-4\
    013de2g3-3196e3g3-8317bcfg3-10870c2fg3-15656bdfg3-6425cdfg3+6477d2fg3-120\
    44befg3-9935cefg3+8451defg3-12443e2fg3-3838bf2g3+10339cf2g3+4743df2g3-253\
    1ef2g3+290f3g3+5711bcg4-11396c2g4+14952bdg4-14504cdg4-6359d2g4-12654beg4-\
    3587ceg4+13324deg4+14021e2g4-4463bfg4+12663cfg4-11006dfg4-7409efg4+823f2g\
    4+7633bg5+3668cg5+14531dg5-11551eg5+14887fg5-10771g6
 ↦ j[58]=e3f3-48cde2fg+8015d2e2fg-6690be3fg-9825ce3fg+10328de3fg-11598e4fg+3\
    317c3f2g+14039bcdf2g-2954c2df2g-562bd2f2g+8290cd2f2g+10421d3f2g-7092bcef2\
    g-7360c2ef2g-4450bdef2g+9748cdef2g+14941d2ef2g-10821be2f2g+626ce2f2g+7182\
    de2f2g+8387e3f2g+15022bcf3g+5428c2f3g-10314bdf3g+7905cdf3g+12520d2f3g-837\
    5bef3g-37cef3g+11467def3g-2053e2f3g-11582bf4g-4904cf4g-7777df4g+2010ef4g+\
    1799f5g-823c4g2-8794bd3g2+11135cd3g2+3332d4g2-13788c3eg2+8209bcdeg2-12819\
    c2deg2-375bd2eg2+12143cd2eg2-13180d3eg2-2406bce2g2+12324c2e2g2+10563bde2g\
    2+6381cde2g2+8962d2e2g2-12361be3g2+15132ce3g2-9489de3g2-2919e4g2+2837c3fg\
    2+8674bcdfg2+11387c2dfg2-14375bd2fg2+4826cd2fg2-10451d3fg2-3442bcefg2+773\
    8c2efg2+14550bdefg2-10011cdefg2+9405d2efg2+11662be2fg2+14409ce2fg2+11669d\
    e2fg2-2860e3fg2+13420bcf2g2-6972c2f2g2-14269bdf2g2-7875cdf2g2+13913d2f2g2\
    +7399bef2g2+3854cef2g2+441def2g2+4207e2f2g2-3843bf3g2-14370cf3g2+4098df3g\
    2+15503ef3g2-8555f4g2+6770c3g3-7463bcdg3+399c2dg3-3856bd2g3+2694cd2g3-713\
    7d3g3+999bceg3+13959c2eg3+11633bdeg3+7129cdeg3+875d2eg3+10552be2g3-6378ce\
    2g3+6892de2g3-5668e3g3-4453bcfg3+2513c2fg3+11487bdfg3+4882cdfg3-7229d2fg3\
    -1898befg3+9860cefg3-14226defg3-10469e2fg3+2416bf2g3-3201cf2g3+14009df2g3\
    +9996ef2g3+5473f3g3+6136bcg4-999c2g4+13936bdg4+7182cdg4-10426d2g4-13807be\
    g4+15661ceg4+10380deg4+8097e2g4+11107bfg4-14646cfg4+9785dfg4+1652efg4+481\
    5f2g4+11427bg5-8888cg5+13570dg5-13004eg5+15964fg5-13681g6
 ↦ j[59]=de2f3+211cde2fg+7965d2e2fg-12834be3fg-15718ce3fg+4198de3fg-5371e4fg\
    +11570c3f2g-12282bcdf2g-8936c2df2g+8053bd2f2g+835cd2f2g+6300d3f2g-5678bce\
    f2g+1662c2ef2g+2767bdef2g-2199cdef2g+9785d2ef2g-15171be2f2g-1122ce2f2g-16\
    65de2f2g-796e3f2g+5600bcf3g+12772c2f3g-1918bdf3g-10568cdf3g-10430d2f3g+14\
    046bef3g-7215cef3g-10720def3g-7085e2f3g-8340bf4g-433cf4g+2875df4g+3970ef4\
    g+10581f5g+501c4g2+98bd3g2+8940cd3g2-15810d4g2+9709c3eg2-2274bcdeg2-2322c\
    2deg2+9459bd2eg2+11356cd2eg2+10249d3eg2-9854bce2g2+12313c2e2g2+507bde2g2-\
    10867cde2g2+7337d2e2g2+2970be3g2-10025ce3g2+15677de3g2+11302e4g2-8395c3fg\
    2+2450bcdfg2-5562c2dfg2-10684bd2fg2-3982cd2fg2-5198d3fg2-7497bcefg2-10352\
    c2efg2-5058bdefg2+2345cdefg2-7926d2efg2-7814be2fg2-5802ce2fg2-1683de2fg2+\
    6545e3fg2-13652bcf2g2-14889c2f2g2+8917bdf2g2+9086cdf2g2+7807d2f2g2+7160be\
    f2g2-6050cef2g2+178def2g2-124e2f2g2-6070bf3g2+11543cf3g2+10514df3g2-10718\
    ef3g2+14711f4g2-3010c3g3+15423bcdg3-2108c2dg3+15246bd2g3+14772cd2g3-1969d\
    3g3-11212bceg3-6778c2eg3-15182bdeg3+2684cdeg3+9674d2eg3-4192be2g3-12501ce\
    2g3+3416de2g3+6731e3g3+734bcfg3+1616c2fg3+10574bdfg3+998cdfg3+4869d2fg3-1\
    834befg3+5605cefg3-320defg3-14075e2fg3+8158bf2g3-4410cf2g3-10321df2g3-327\
    5ef2g3-9443f3g3-3987bcg4+9831c2g4-1431bdg4+9127cdg4-13096d2g4+13711beg4+7\
    829ceg4+11329deg4-3663e2g4-4811bfg4+14028cfg4+14305dfg4-9069efg4+13274f2g\
    4+3899bg5+14657cg5-4050dg5-3675eg5+1906fg5-1757g6
 ↦ j[60]=ce2f3-5987cde2fg-3474d2e2fg-11935be3fg+12384ce3fg-12962de3fg-4192e4\
    fg+8406c3f2g-12129bcdf2g-670c2df2g+9695bd2f2g-7269cd2f2g-9824d3f2g-11234b\
    cef2g-32c2ef2g+12146bdef2g-2823cdef2g+13430d2ef2g+2579be2f2g-2959ce2f2g+3\
    802de2f2g+4835e3f2g+2007bcf3g+5314c2f3g-4322bdf3g+15163cdf3g-15272d2f3g+9\
    510bef3g+9999cef3g-7035def3g+11640e2f3g-9575bf4g+3549cf4g+640df4g+2616ef4\
    g+2081f5g-7896c4g2-7343bd3g2-1632cd3g2+9461d4g2-7965c3eg2-13765bcdeg2-943\
    6c2deg2+2869bd2eg2-625cd2eg2+1504d3eg2-10057bce2g2+5310c2e2g2-10757bde2g2\
    -410cde2g2-15840d2e2g2+8195be3g2+2363ce3g2+6549de3g2+14885e4g2+6905c3fg2-\
    2751bcdfg2+3517c2dfg2-5469bd2fg2+10217cd2fg2+1614d3fg2-1860bcefg2+12610c2\
    efg2+1131bdefg2-9803cdefg2-15653d2efg2+13393be2fg2+10850ce2fg2-12960de2fg\
    2-5761e3fg2-3659bcf2g2-6532c2f2g2-6047bdf2g2-4436cdf2g2-2422d2f2g2+7410be\
    f2g2+5272cef2g2-14349def2g2-15685e2f2g2+12303bf3g2-10830cf3g2+15475df3g2-\
    14424ef3g2-11589f4g2+7093c3g3+6937bcdg3-8268c2dg3+10206bd2g3+2698cd2g3-10\
    510d3g3-8488bceg3+6404c2eg3+5728bdeg3-11401cdeg3+13770d2eg3+8890be2g3+804\
    4ce2g3+7828de2g3+14796e3g3+8386bcfg3+1644c2fg3-4622bdfg3+13351cdfg3+10869\
    d2fg3-14142befg3+13132cefg3+13684defg3-5311e2fg3-9661bf2g3+14863cf2g3+270\
    5df2g3+2513ef2g3-14441f3g3-5100bcg4-5913c2g4+4564bdg4-2434cdg4+2384d2g4-1\
    5201beg4+7378ceg4-2687deg4+15027e2g4-7197bfg4-2263cfg4+4577dfg4-2536efg4-\
    369f2g4-14641bg5-1398cg5-7569dg5+7879eg5-6098fg5+14920g6
 ↦ j[61]=be2f3-6799cde2fg+3368d2e2fg-5562be3fg-15778ce3fg+14315de3fg-6308e4f\
    g+2471c3f2g+3591bcdf2g-2815c2df2g+3760bd2f2g-8841cd2f2g-825d3f2g-4922bcef\
    2g+11715c2ef2g+15865bdef2g+4345cdef2g+2380d2ef2g+574be2f2g-8138ce2f2g-120\
    13de2f2g-10401e3f2g+11169bcf3g+15382c2f3g-11968bdf3g-9333cdf3g-5899d2f3g-\
    8770bef3g+3536cef3g-4958def3g-1276e2f3g+8898bf4g+10395cf4g-9103df4g+15543\
    ef4g+13580f5g-15167c4g2-9753bd3g2-9793cd3g2+15029d4g2-5559c3eg2-10808bcde\
    g2+14348c2deg2+12786bd2eg2-3522cd2eg2+13696d3eg2+8459bce2g2-12372c2e2g2+1\
    001bde2g2+2654cde2g2-3449d2e2g2+615be3g2-1296ce3g2-5286de3g2+4075e4g2+131\
    46c3fg2-5156bcdfg2+8483c2dfg2+11558bd2fg2+10954cd2fg2-4884d3fg2-5405bcefg\
    2+3859c2efg2+1073bdefg2+10628cdefg2-11738d2efg2-3541be2fg2-5450ce2fg2-113\
    de2fg2-13691e3fg2+3877bcf2g2-5818c2f2g2+4012bdf2g2-12603cdf2g2-247d2f2g2+\
    1106bef2g2-896cef2g2-12808def2g2+7283e2f2g2-10325bf3g2+9919cf3g2-8506df3g\
    2+2052ef3g2+12936f4g2-1194c3g3+5469bcdg3+12201c2dg3+5272bd2g3-6449cd2g3+5\
    327d3g3-11226bceg3+13856c2eg3+8675bdeg3+12636cdeg3-8581d2eg3+2330be2g3+14\
    064ce2g3+14021de2g3+4282e3g3+940bcfg3-6664c2fg3-13068bdfg3-8686cdfg3-1328\
    7d2fg3-7769befg3+10781cefg3-5232defg3+1070e2fg3+1024bf2g3-7984cf2g3-6199d\
    f2g3+868ef2g3+8030f3g3-15715bcg4+10870c2g4+837bdg4+15498cdg4-14000d2g4+13\
   482beg4-14566ceg4+6270deg4+11629e2g4-6633bfg4+6391cfg4+14006dfg4-13071efg\
   4-6857f2g4+4247bg5-13506cg5+1913dg5-13074eg5+11735fg5+5498g6
 ↦ j[62]=d2ef3-4369cde2fg-13653d2e2fg-11376be3fg-337ce3fg-13703de3fg+2609e4f\
   g-11037c3f2g-10692bcdf2g+10819c2df2g+12605bd2f2g+11784cd2f2g+14073d3f2g+4\
   114bcef2g-8691c2ef2g-11164bdef2g-214cdef2g+2671d2ef2g-4352be2f2g+11359ce2\
   f2g-3694de2f2g-7458e3f2g+1336bcf3g-15596c2f3g+5316bdf3g-142cdf3g+925d2f3g\
   +6050bef3g-15985cef3g+9853def3g-2517e2f3g+14319bf4g-6754cf4g-12427df4g-10\
   891ef4g-692f5g-3007c4g2+15208bd3g2-2287cd3g2+11609d4g2-505c3eg2-14036bcde\
   g2-11484c2deg2-8638bd2eg2+5947cd2eg2+5202d3eg2-1082bce2g2-9132c2e2g2-7801\
   bde2g2+10738cde2g2-7841d2e2g2-4344be3g2+10009ce3g2-15551de3g2-10473e4g2+1\
   4744c3fg2-630bcdfg2+13066c2dfg2+15566bd2fg2+13744cd2fg2-3056d3fg2-14857bc\
   efg2+15172c2efg2+6019bdefg2-15311cdefg2-3234d2efg2+11888be2fg2+594ce2fg2+\
   1071de2fg2-12787e3fg2+13729bcf2g2-15234c2f2g2+2923bdf2g2+15794cdf2g2+2825\
   d2f2g2+2862bef2g2+15021cef2g2-1146def2g2+14546e2f2g2+3967bf3g2+10425cf3g2\
   -1766df3g2+15640ef3g2+2086f4g2+7126c3g3-6565bcdg3-10232c2dg3+13267bd2g3-1\
   176cd2g3-10798d3g3+6014bceg3+10695c2eg3+219bdeg3-14964cdeg3+12345d2eg3+56\
   85be2g3+15516ce2g3+2502de2g3-8674e3g3-10155bcfg3-4340c2fg3+4757bdfg3-1391\
   1cdfg3+4497d2fg3+6353befg3+14903cefg3+14122defg3-12998e2fg3+2680bf2g3-131\
   04cf2g3-12167df2g3-10626ef2g3+12405f3g3+3072bcg4+648c2g4-11026bdg4-11701c\
   dg4-5779d2g4-940beg4+373ceg4-8021deg4-8148e2g4-6988bfg4-4944cfg4+6556dfg4\
   +372efg4+2841f2g4-1300bg5-12412cg5-1973dg5-13888eg5+14026fg5-4430g6
 ↦ j[63]=cdef3+15812cde2fg-5672d2e2fg-5772be3fg+8019ce3fg+5487de3fg+9188e4fg\
   +15788c3f2g+11081bcdf2g+293c2df2g-14430bd2f2g-6641cd2f2g+10464d3f2g+14382\
   bcef2g+6739c2ef2g+1160bdef2g+13060cdef2g+1539d2ef2g-12267be2f2g+9898ce2f2\
   g+7546de2f2g-10115e3f2g+2245bcf3g-2625c2f3g+9838bdf3g+3339cdf3g-10921d2f3\
   g+15597bef3g-6945cef3g-7973def3g+7152e2f3g+7641bf4g-11836cf4g+2502df4g-10\
   632ef4g-4727f5g-7396c4g2-6578bd3g2-2962cd3g2+9222d4g2-10441c3eg2+4574bcde\
   g2+10937c2deg2-3944bd2eg2-2558cd2eg2-7503d3eg2+4170bce2g2-644c2e2g2+13249\
   bde2g2+1581cde2g2-11129d2e2g2+13778be3g2+15785ce3g2+4320de3g2-14829e4g2-7\
   582c3fg2-9795bcdfg2-8034c2dfg2+10217bd2fg2+5539cd2fg2-936d3fg2-894bcefg2+\
   6139c2efg2-15082bdefg2-12416cdefg2-11071d2efg2+4266be2fg2-11555ce2fg2+913\
   9de2fg2-4768e3fg2+1733bcf2g2-11927c2f2g2-2446bdf2g2+4605cdf2g2-8563d2f2g2\
   -5910bef2g2+4960cef2g2+8935def2g2-8617e2f2g2+295bf3g2-15940cf3g2-4525df3g\
   2+13374ef3g2+4955f4g2-6569c3g3-9275bcdg3+3939c2dg3+345bd2g3-1052cd2g3+926\
   4d3g3-15113bceg3+9504c2eg3-14744bdeg3+5913cdeg3+967d2eg3+14589be2g3-1728c\
   e2g3+15988de2g3+9161e3g3+5399bcfg3-11613c2fg3+15729bdfg3-9150cdfg3+5981d2\
   fg3-13799befg3+15359cefg3-1523defg3-3961e2fg3+14470bf2g3+9165cf2g3+11839d\
   f2g3-12787ef2g3-222f3g3-7315bcg4+15179c2g4+352bdg4+3456cdg4-9696d2g4-357b\
   eg4+6252ceg4-5291deg4-15158e2g4-12657bfg4-14760cfg4+6965dfg4+2402efg4+115\
   14f2g4-14732bg5-148cg5+13659dg5+14420eg5+15098fg5-3196g6
 ↦ j[64]=bdef3+13782cde2fg+15475d2e2fg-2520be3fg+15306ce3fg+15795de3fg-9167e\
   4fg+11114c3f2g+7238bcdf2g-12855c2df2g+8032bd2f2g+8218cd2f2g+2048d3f2g-157\
   50bcef2g+26c2ef2g+12195bdef2g-8918cdef2g+8543d2ef2g-6346be2f2g-13509ce2f2\
   g+7899de2f2g-7136e3f2g+1846bcf3g-13865c2f3g+7024bdf3g-9591cdf3g-4435d2f3g\
   +336bef3g+14054cef3g+9292def3g+13605e2f3g-7781bf4g-11781cf4g+14242df4g+13\
   091ef4g+6928f5g-14722c4g2+5825bd3g2+9575cd3g2+12221d4g2-7448c3eg2-7281bcd\
   eg2-248c2deg2+1677bd2eg2+11968cd2eg2+3716d3eg2+6438bce2g2-12451c2e2g2+568\
   9bde2g2+12321cde2g2+6563d2e2g2+14947be3g2-14868ce3g2+1558de3g2-3189e4g2-1\
   0761c3fg2+682bcdfg2+8324c2dfg2-1186bd2fg2-12436cd2fg2-15846d3fg2-14820bce\
   fg2-1504c2efg2+14761bdefg2+15030cdefg2-1628d2efg2-3653be2fg2-95ce2fg2+801\
   6de2fg2+2029e3fg2+1525bcf2g2+12908c2f2g2-11361bdf2g2+3744cdf2g2+9029d2f2g\
   2+10223bef2g2-2821cef2g2+4414def2g2-6802e2f2g2+7559bf3g2-4148cf3g2+3708df\
   3g2+5408ef3g2+11154f4g2-5328c3g3-6850bcdg3-8472c2dg3+2486bd2g3+532cd2g3-4\
      935d3g3-12732bceg3+6307c2eg3+2952bdeg3-7070cdeg3+7680d2eg3-4486be2g3+9106\
      ce2g3-15589de2g3-5857e3g3+8894bcfg3+1970c2fg3+13475bdfg3+14986cdfg3+2385d\
      2fg3-6233befg3-15127cefg3-13225defg3-8481e2fg3+931bf2g3+2623cf2g3-7932df2\
      g3+11030ef2g3-5124f3g3-12208bcg4-2431c2g4-11821bdg4+12151cdg4+13988d2g4+1\
      5081beg4+13132ceg4-10729deg4+3398e2g4-7449bfg4+1032cfg4-2545dfg4-2992efg4\
      +10449f2g4+14663bg5+15633cg5-916dg5+3155eg5+11747fg5-7419g6
 ↦ j[65]=c2ef3+2024cde2fg-15991d2e2fg+10969be3fg-13223ce3fg+11164de3fg-10384\
      e4fg+3649c3f2g-5bcdf2g-10141c2df2g+8143bd2f2g+12328cd2f2g-1923d3f2g-14514\
      bcef2g+15602c2ef2g+9450bdef2g-8248cdef2g-3351d2ef2g-7815be2f2g-15063ce2f2\
      g+13441de2f2g+1121e3f2g+108bcf3g+2027c2f3g-12049bdf3g-9806cdf3g+9528d2f3g\
      +837bef3g-13530cef3g+5599def3g-12840e2f3g+8124bf4g-7398cf4g-9316df4g-1868\
      ef4g+9773f5g+4856c4g2+15931bd3g2-14293cd3g2-15714d4g2+2732c3eg2+4161bcdeg\
      2+12724c2deg2-8140bd2eg2-15990cd2eg2+9089d3eg2+3740bce2g2-8947c2e2g2+1095\
      4bde2g2-14757cde2g2-5649d2e2g2-3958be3g2-12202ce3g2-6785de3g2+10406e4g2-5\
      626c3fg2+9072bcdfg2+15203c2dfg2+13493bd2fg2+1543cd2fg2+8431d3fg2-1355bcef\
      g2-8719c2efg2-14289bdefg2-13613cdefg2-11927d2efg2-10157be2fg2+4131ce2fg2-\
      459de2fg2+15877e3fg2-7725bcf2g2+9420c2f2g2-13938bdf2g2+2459cdf2g2+6271d2f\
      2g2-11914bef2g2-3801cef2g2-13777def2g2-15884e2f2g2+10001bf3g2+7599cf3g2-9\
      687df3g2+11690ef3g2+15116f4g2+2589c3g3+1924bcdg3-2741c2dg3+15481bd2g3-119\
      36cd2g3+1859d3g3+6915bceg3-12627c2eg3+62bdeg3-4436cdeg3+13612d2eg3+10340b\
      e2g3-4856ce2g3-6682de2g3+14112e3g3+12538bcfg3+4465c2fg3-6910bdfg3+4121cdf\
      g3-13432d2fg3-13254befg3-88cefg3-14938defg3+3082e2fg3-12881bf2g3-863cf2g3\
      -3430df2g3-14351ef2g3+4960f3g3+3654bcg4+14511c2g4-12656bdg4+7659cdg4-7478\
      d2g4-5913beg4-3276ceg4+11948deg4-1997e2g4-5090bfg4+3642cfg4+14244dfg4+958\
      4efg4+1800f2g4+6666bg5-635cg5-12660dg5+1229eg5+8785fg5-14677g6
 ↦ j[66]=bcef3+749cde2fg+12709d2e2fg-9248be3fg+4522ce3fg-1558de3fg+12592e4fg\
      +3726c3f2g-2602bcdf2g-1981c2df2g+8872bd2f2g+4534cd2f2g+1506d3f2g+15469bce\
      f2g+1067c2ef2g-6988bdef2g-4695cdef2g+13378d2ef2g+6651be2f2g+15782ce2f2g+1\
      1426de2f2g-8509e3f2g-15415bcf3g-7690c2f3g+5462bdf3g+14737cdf3g+13090d2f3g\
      -10982bef3g+2750cef3g-2948def3g-9434e2f3g+15831bf4g-1518cf4g-1946df4g+666\
      5ef4g-4659f5g+4070c4g2+5828bd3g2-11367cd3g2-5862d4g2+5894c3eg2+10687bcdeg\
      2-3331c2deg2+1843bd2eg2+6086cd2eg2-6271d3eg2-1896bce2g2-8848c2e2g2-14671b\
      de2g2-14869cde2g2+6344d2e2g2+5857be3g2+14630ce3g2-4318de3g2-2192e4g2+1404\
      2c3fg2+4034bcdfg2-7419c2dfg2-9030bd2fg2+10804cd2fg2+14406d3fg2+6255bcefg2\
      -13601c2efg2-3876bdefg2-10359cdefg2-11567d2efg2+15226be2fg2+12049ce2fg2+5\
      03de2fg2-3287e3fg2-15344bcf2g2-2665c2f2g2+5163bdf2g2+248cdf2g2-2286d2f2g2\
      -13042bef2g2-2677cef2g2-3480def2g2-3763e2f2g2+11082bf3g2-4462cf3g2+9300df\
      3g2-4682ef3g2+15667f4g2-2251c3g3+12575bcdg3-14216c2dg3+6600bd2g3+7094cd2g\
      3-4072d3g3-7320bceg3-3177c2eg3+12972bdeg3-7901cdeg3-12676d2eg3+14402be2g3\
      -5556ce2g3+12321de2g3-4253e3g3-13286bcfg3-13220c2fg3-7675bdfg3+8449cdfg3+\
      15374d2fg3+4875befg3-6837cefg3-1280defg3+3069e2fg3+13432bf2g3-10009cf2g3+\
      4484df2g3+7600ef2g3-12653f3g3-5441bcg4+3036c2g4+13869bdg4-2921cdg4+15818d\
      2g4+11814beg4+3762ceg4-4825deg4-8555e2g4-9922bfg4-10871cfg4+8930dfg4-2640\
      efg4-3067f2g4+1944bg5+10129cg5+1594dg5+15848eg5-3452fg5-12105g6
 ↦ j[67]=d3f3-10887cde2fg+2813d2e2fg-13931be3fg-1780ce3fg+2881de3fg+11150e4f\
      g-2679c3f2g+13492bcdf2g-3277c2df2g+3331bd2f2g+10865cd2f2g-7211d3f2g+10546\
      bcef2g+10321c2ef2g+15208bdef2g+5910cdef2g+9362d2ef2g-617be2f2g+1168ce2f2g\
      -10534de2f2g+10532e3f2g+10625bcf3g-12852c2f3g+13080bdf3g-6320cdf3g-10320d\
      2f3g-5676bef3g+7583cef3g+2754def3g-7075e2f3g-13616bf4g+10376cf4g+12088df4\
      g+436ef4g-2236f5g-1483c4g2+11086bd3g2-2526cd3g2+6799d4g2+1601c3eg2-1337bc\
      deg2+14875c2deg2-12304bd2eg2-365cd2eg2+12893d3eg2+3404bce2g2+4478c2e2g2-7\
      513bde2g2+9868cde2g2+994d2e2g2-10022be3g2+7304ce3g2+5983de3g2+7915e4g2-27\
      35c3fg2+3047bcdfg2+12458c2dfg2+1016bd2fg2+1545cd2fg2-12896d3fg2+2153bcefg\
    2-152c2efg2+14952bdefg2+11391cdefg2-15466d2efg2+14420be2fg2+9078ce2fg2+13\
    280de2fg2-1078e3fg2-8304bcf2g2-9055c2f2g2+7329bdf2g2-6136cdf2g2-6077d2f2g\
    2+14521bef2g2+8113cef2g2-13162def2g2-13455e2f2g2+8050bf3g2-14395cf3g2+265\
    8df3g2-15048ef3g2-2187f4g2+8647c3g3+5623bcdg3+8220c2dg3-2762bd2g3-808cd2g\
    3+8820d3g3-15124bceg3+13723c2eg3+10933bdeg3+15165cdeg3+6146d2eg3+8366be2g\
    3-9404ce2g3-933de2g3-4754e3g3+15910bcfg3+10977c2fg3+407bdfg3-9059cdfg3+29\
    2d2fg3+12587befg3+5660cefg3+103defg3-12355e2fg3+11851bf2g3-12141cf2g3-779\
    3df2g3+13001ef2g3-9075f3g3-2733bcg4+4579c2g4+1216bdg4+186cdg4+5286d2g4-15\
    419beg4-14329ceg4-7942deg4+11859e2g4-10168bfg4-12506cfg4-9189dfg4-12674ef\
    g4+997f2g4+2382bg5+8062cg5-6908dg5-756eg5+2998fg5-178g6
 ↦ j[68]=cd2f3-6758cde2fg+12383d2e2fg+134be3fg-13998ce3fg-11555de3fg-1300e4f\
    g-12623c3f2g-8785bcdf2g+14388c2df2g-11737bd2f2g-8675cd2f2g-9677d3f2g+1529\
    9bcef2g-2807c2ef2g+12017bdef2g-1974cdef2g+5421d2ef2g+3349be2f2g-10408ce2f\
    2g-11702de2f2g-10658e3f2g-15244bcf3g+604c2f3g+1136bdf3g+4520cdf3g-12173d2\
    f3g-3189bef3g+5265cef3g-12995def3g-11538e2f3g+548bf4g+5275cf4g+11899df4g+\
    4510ef4g+1567f5g-862c4g2-5650bd3g2-13242cd3g2-6334d4g2-14271c3eg2-1931bcd\
    eg2-5841c2deg2+7159bd2eg2+13873cd2eg2+14514d3eg2+1790bce2g2+12615c2e2g2-1\
    2806bde2g2+12056cde2g2-1556d2e2g2+6903be3g2+10520ce3g2+14648de3g2+14541e4\
    g2+11092c3fg2-3323bcdfg2+7098c2dfg2+15944bd2fg2+9041cd2fg2-8990d3fg2+1591\
    0bcefg2-14443c2efg2+1753bdefg2-5708cdefg2+2065d2efg2+8857be2fg2+3133ce2fg\
    2-10930de2fg2-2705e3fg2-3622bcf2g2+13681c2f2g2+15544bdf2g2-13097cdf2g2+38\
    5d2f2g2-4058bef2g2-2866cef2g2+1885def2g2-4373e2f2g2+2847bf3g2+14585cf3g2-\
    15856df3g2+10272ef3g2-6354f4g2-2892c3g3-14959bcdg3-11544c2dg3-3956bd2g3-6\
    905cd2g3+959d3g3-9110bceg3-12948c2eg3-4530bdeg3+12052cdeg3-14107d2eg3-816\
    8be2g3-14809ce2g3+9737de2g3+8093e3g3+14015bcfg3+4894c2fg3-2715bdfg3+7709c\
    dfg3+8109d2fg3-3784befg3+5095cefg3+9852defg3+9571e2fg3-11096bf2g3+11782cf\
    2g3+14902df2g3+6470ef2g3+5144f3g3+2759bcg4-6428c2g4+1572bdg4+4533cdg4-227\
    3d2g4+14510beg4-4571ceg4+7107deg4-2711e2g4+9253bfg4+4104cfg4-3329dfg4+687\
    4efg4+15900f2g4+9367bg5+12509cg5+13216dg5-8148eg5+12245fg5-1754g6
 ↦ j[69]=bd2f3+3416cde2fg-15793d2e2fg+11108be3fg-10174ce3fg+12109de3fg+11767\
    e4fg+13326c3f2g+8883bcdf2g-3083c2df2g-3682bd2f2g-10073cd2f2g+179d3f2g-259\
    bcef2g-8606c2ef2g+14727bdef2g+286cdef2g+4905d2ef2g+202be2f2g-14545ce2f2g+\
    3836de2f2g+1245e3f2g-11504bcf3g-8914c2f3g+10956bdf3g+5146cdf3g-9974d2f3g-\
    2817bef3g+11216cef3g+15015def3g+6299e2f3g+15382bf4g-7391cf4g+2585df4g+145\
    26ef4g-8265f5g-10311c4g2+5168bd3g2-4446cd3g2-2219d4g2-9414c3eg2+4361bcdeg\
    2+12929c2deg2+101bd2eg2-10778cd2eg2+9744d3eg2-1158bce2g2-11870c2e2g2-8764\
    bde2g2-15056cde2g2+7745d2e2g2-9987be3g2-9868ce3g2-15992de3g2-8677e4g2-430\
    4c3fg2+12131bcdfg2+13165c2dfg2-12976bd2fg2+5431cd2fg2+9411d3fg2-14525bcef\
    g2-14593c2efg2-4651bdefg2-9028cdefg2+4525d2efg2-1307be2fg2-2281ce2fg2+142\
    13de2fg2+7363e3fg2-13756bcf2g2-2677c2f2g2-5876bdf2g2+11342cdf2g2-14760d2f\
    2g2-10543bef2g2+2168cef2g2-3485def2g2+7686e2f2g2+12612bf3g2-13977cf3g2+70\
    20df3g2+8126ef3g2-8803f4g2+6559c3g3-9271bcdg3+3346c2dg3+818bd2g3+12240cd2\
    g3-831d3g3+697bceg3-7296c2eg3+10554bdeg3-13151cdeg3-6393d2eg3+12005be2g3-\
    11504ce2g3-12178de2g3+7648e3g3+8817bcfg3+2063c2fg3-9597bdfg3-12806cdfg3+1\
    0523d2fg3-3168befg3-11321cefg3-15594defg3-6011e2fg3+15686bf2g3+13472cf2g3\
    +3455df2g3-6424ef2g3+9685f3g3-8021bcg4+8798c2g4+15282bdg4+208cdg4-14881d2\
    g4+3812beg4+1449ceg4+11737deg4+3436e2g4+2448bfg4+12452cfg4+13212dfg4+1551\
    6efg4+13596f2g4-1886bg5-4162cg5-10711dg5-5031eg5-4634fg5+4163g6
 ↦ j[70]=c2df3+7496cde2fg-15575d2e2fg+2881be3fg+4928ce3fg+12129de3fg-1870e4f\
    g-2209c3f2g+2202bcdf2g-1736c2df2g-10203bd2f2g+4148cd2f2g+10715d3f2g-8290b\
    cef2g-7850c2ef2g-7713bdef2g-7926cdef2g-4112d2ef2g-12052be2f2g+6660ce2f2g-\
    9114de2f2g-2317e3f2g-1080bcf3g+15645c2f3g-7536bdf3g-15146cdf3g-3520d2f3g-\
    9798bef3g+7415cef3g+14217def3g-7237e2f3g-528bf4g-5236cf4g-14173df4g+7264e\
    f4g-1041f5g+5703c4g2-3969bd3g2+645cd3g2-4635d4g2-4407c3eg2+12775bcdeg2-65\
    38c2deg2+4227bd2eg2-1583cd2eg2-10054d3eg2-8961bce2g2-8480c2e2g2-2577bde2g\
    2-7628cde2g2-3580d2e2g2+2027be3g2+5542ce3g2+4582de3g2-2214e4g2-10932c3fg2\
    +4504bcdfg2+1713c2dfg2-13334bd2fg2+5092cd2fg2-10799d3fg2+8064bcefg2-3582c\
    2efg2-11bdefg2+10289cdefg2-12083d2efg2+10909be2fg2+9543ce2fg2+9297de2fg2-\
    11228e3fg2-14551bcf2g2-7263c2f2g2-7209bdf2g2+8163cdf2g2-13380d2f2g2+12409\
    bef2g2+2951cef2g2+11525def2g2+3908e2f2g2-6475bf3g2+11858cf3g2-14140df3g2+\
    11295ef3g2-8804f4g2+10990c3g3+15867bcdg3+14924c2dg3+13776bd2g3-14982cd2g3\
    -15887d3g3+2051bceg3+9451c2eg3-10342bdeg3-9365cdeg3+2950d2eg3-15206be2g3-\
    6596ce2g3+545de2g3-2114e3g3-5423bcfg3+9694c2fg3+3276bdfg3-9202cdfg3-6911d\
    2fg3-1504befg3-3916cefg3+1887defg3+7474e2fg3-8245bf2g3-6161cf2g3+1608df2g\
    3+833ef2g3-4934f3g3-7584bcg4+3318c2g4-3019bdg4-2699cdg4-4920d2g4+10974beg\
    4+13146ceg4-11358deg4+1802e2g4+6411bfg4-1136cfg4+8162dfg4+11045efg4+4558f\
    2g4-10749bg5+7764cg5-4440dg5+9814eg5-14793fg5+15114g6
 ↦ j[71]=bcdf3+1343cde2fg-5392d2e2fg-13644be3fg+5869ce3fg-13161de3fg-16001e4\
    fg+4999c3f2g-4959bcdf2g-5263c2df2g-12723bd2f2g-10283cd2f2g-14523d3f2g+120\
    10bcef2g+12046c2ef2g-2423bdef2g+11308cdef2g-9296d2ef2g+12908be2f2g+12042c\
    e2f2g+3400de2f2g+13727e3f2g+14676bcf3g-13729c2f3g-2693bdf3g-6810cdf3g+436\
    0d2f3g+6645bef3g-6034cef3g-3314def3g+11850e2f3g-14452bf4g+9065cf4g-11608d\
    f4g-8121ef4g-266f5g+3447c4g2+10944bd3g2-6561cd3g2-7337d4g2+15533c3eg2-362\
    3bcdeg2-11221c2deg2-6781bd2eg2+257cd2eg2+5267d3eg2+9067bce2g2+5501c2e2g2+\
    11741bde2g2-9213cde2g2+5403d2e2g2+15843be3g2-10567ce3g2-12122de3g2-5275e4\
    g2+12702c3fg2+8837bcdfg2-13997c2dfg2+10734bd2fg2-4209cd2fg2-14169d3fg2-78\
    96bcefg2+289c2efg2-4351bdefg2-13202cdefg2-6840d2efg2+2276be2fg2-38ce2fg2+\
    9037de2fg2-894e3fg2-13115bcf2g2+12038c2f2g2-10997bdf2g2-4242cdf2g2+7765d2\
    f2g2-13250bef2g2+11058cef2g2-9013def2g2-4490e2f2g2+8608bf3g2+10832cf3g2+2\
    906df3g2-94ef3g2-8851f4g2-4245c3g3+6603bcdg3-10735c2dg3-4731bd2g3-8216cd2\
    g3-8509d3g3-4272bceg3-3231c2eg3+3984bdeg3-10508cdeg3+2006d2eg3-11493be2g3\
    -9970ce2g3-2489de2g3+336e3g3+8179bcfg3-508c2fg3+5315bdfg3-12290cdfg3-1424\
    1d2fg3+6287befg3+5166cefg3+10362defg3-15036e2fg3-794bf2g3-2995cf2g3+9336d\
    f2g3-7736ef2g3-11776f3g3-14123bcg4-8798c2g4+2349bdg4+14681cdg4-13338d2g4+\
    7507beg4-13274ceg4-9409deg4-13352e2g4-3456bfg4-4872cfg4+463dfg4-9975efg4+\
    1459f2g4-14738bg5+949cg5-3173dg5+4130eg5+1747fg5-1888g6
 ↦ j[72]=c3f3-11785cde2fg-15754d2e2fg+4393be3fg+4747ce3fg+12107de3fg+3105e4f\
    g+14338c3f2g+14829bcdf2g+9522c2df2g-2614bd2f2g-10297cd2f2g-7886d3f2g-1107\
    8bcef2g+10460c2ef2g-9233bdef2g-937cdef2g-4639d2ef2g-7992be2f2g-2820ce2f2g\
    +12215de2f2g-5948e3f2g-2632bcf3g-1878c2f3g+7180bdf3g-12450cdf3g+8064d2f3g\
    -9463bef3g-5139cef3g+11179def3g+4693e2f3g+5285bf4g+6565cf4g-1214df4g+8756\
    ef4g-11418f5g-8314c4g2+2864bd3g2-13318cd3g2+6109d4g2+9853c3eg2+5565bcdeg2\
    -3521c2deg2-13983bd2eg2+6954cd2eg2+6749d3eg2-14182bce2g2-8528c2e2g2-5036b\
    de2g2-6853cde2g2-5018d2e2g2+9881be3g2+7512ce3g2-13300de3g2-384e4g2+15317c\
    3fg2-4101bcdfg2+7597c2dfg2+5001bd2fg2+7679cd2fg2-637d3fg2+5284bcefg2-1774\
    c2efg2+9343bdefg2+15134cdefg2+14709d2efg2+5752be2fg2-9938ce2fg2-10059de2f\
    g2+11334e3fg2+3103bcf2g2-1549c2f2g2-8880bdf2g2+4520cdf2g2+1654d2f2g2+248b\
    ef2g2+11366cef2g2+11094def2g2+373e2f2g2-9989bf3g2+13196cf3g2+7847df3g2+16\
    05ef3g2+2275f4g2+11666c3g3+15297bcdg3+13170c2dg3+9462bd2g3+12189cd2g3-532\
    6d3g3-2036bceg3+9103c2eg3+12255bdeg3+6851cdeg3+13353d2eg3-3830be2g3+12607\
    ce2g3-6015de2g3-9567e3g3-10614bcfg3-5085c2fg3+8748bdfg3+9388cdfg3-13824d2\
    fg3+8646befg3+5742cefg3-868defg3-10695e2fg3-12102bf2g3+10554cf2g3+15448df\
    2g3-15522ef2g3+1224f3g3+10944bcg4+1809c2g4-13433bdg4+12902cdg4+9152d2g4-6\
    106beg4+8316ceg4-8970deg4-12313e2g4+3395bfg4-11049cfg4+6853dfg4-7807efg4-\
    5477f2g4-171bg5-7497cg5-11399dg5+13723eg5-4752fg5-3143g6
 ↦ j[73]=e4f2+15908cde2fg-13642d2e2fg+15201be3fg+9371ce3fg-5499de3fg-4149e4f\
      g+9012c3f2g+12725bcdf2g+6682c2df2g-477bd2f2g+11243cd2f2g+7845d3f2g+13621b\
      cef2g+5522c2ef2g+5150bdef2g-7476cdef2g-10465d2ef2g-10720be2f2g-13814ce2f2\
      g+3885de2f2g-13858e3f2g-8197bcf3g+1702c2f3g-6609bdf3g+8343cdf3g+3909d2f3g\
      -7305bef3g+5618cef3g+10600def3g-15300e2f3g-3045bf4g+10067cf4g-1679df4g-15\
      673ef4g-14763f5g+6316c4g2+12133bd3g2-990cd3g2-4504d4g2-8234c3eg2+6671bcde\
      g2+13533c2deg2-4033bd2eg2+15326cd2eg2-4630d3eg2+6662bce2g2+4135c2e2g2+138\
      3bde2g2-2568cde2g2-9589d2e2g2-4366be3g2+913ce3g2+6029de3g2+13405e4g2+7836\
      c3fg2-6971bcdfg2-6549c2dfg2-4943bd2fg2+257cd2fg2+2263d3fg2-15735bcefg2+95\
      21c2efg2-3827bdefg2-13968cdefg2-4502d2efg2-10630be2fg2+5551ce2fg2-14233de\
      2fg2+3800e3fg2+6306bcf2g2-8244c2f2g2-12148bdf2g2+817cdf2g2-13853d2f2g2+12\
      552bef2g2-691cef2g2+14363def2g2-3379e2f2g2-3834bf3g2-9875cf3g2+5264df3g2+\
      770ef3g2+1694f4g2+11063c3g3-7186bcdg3-3941c2dg3-9088bd2g3-14313cd2g3+876d\
      3g3-6228bceg3+11777c2eg3+4007bdeg3+1180cdeg3-11313d2eg3-11799be2g3-8236ce\
      2g3+534de2g3+15473e3g3+11136bcfg3+14713c2fg3-3069bdfg3+15779cdfg3-11499d2\
      fg3+11318befg3+8386cefg3-13583defg3-7739e2fg3-10257bf2g3+1880cf2g3-14590d\
      f2g3-1388ef2g3-15942f3g3+13206bcg4-6447c2g4+12761bdg4+2436cdg4-6930d2g4+1\
      4478beg4+13974ceg4+3057deg4+6803e2g4-886bfg4+3405cfg4+1438dfg4+12669efg4-\
      4100f2g4-7957bg5-2457cg5+6802dg5-11985eg5-1465fg5+4309g6
  ↦ j[74]=de3f2-14944cde2fg-8960d2e2fg-13333be3fg+9437ce3fg+1601de3fg+9752e4f\
      g-14993c3f2g+3840bcdf2g-10357c2df2g-15059bd2f2g+11584cd2f2g-3418d3f2g-794\
      5bcef2g-15346c2ef2g+15298bdef2g-10521cdef2g-2487d2ef2g+1316be2f2g+10151ce\
      2f2g+10833de2f2g-3496e3f2g+5532bcf3g-10518c2f3g-6827bdf3g+8071cdf3g-3210d\
      2f3g+1720bef3g+9725cef3g+15503def3g-2956e2f3g-12614bf4g-7328cf4g-11103df4\
      g+15541ef4g-3535f5g-6244c4g2+9008bd3g2-9035cd3g2+12405d4g2+13827c3eg2-745\
      6bcdeg2-2899c2deg2-3007bd2eg2+13288cd2eg2-4744d3eg2-7187bce2g2+6723c2e2g2\
      +6337bde2g2-12071cde2g2-8243d2e2g2-15763be3g2-11432ce3g2+5708de3g2-10662e\
      4g2-11747c3fg2-1030bcdfg2-4898c2dfg2-5082bd2fg2-13539cd2fg2+1323d3fg2+131\
      25bcefg2-9701c2efg2-9045bdefg2-13921cdefg2-4778d2efg2-7356be2fg2-5864ce2f\
      g2+15941de2fg2+6998e3fg2+1997bcf2g2-15596c2f2g2-13479bdf2g2-11353cdf2g2+4\
      991d2f2g2-11687bef2g2+15812cef2g2-1847def2g2+10854e2f2g2+7698bf3g2-8291cf\
      3g2+10485df3g2+7166ef3g2+1630f4g2+14686c3g3-186bcdg3+4959c2dg3+6463bd2g3+\
      7275cd2g3+1681d3g3+10493bceg3-15148c2eg3+10500bdeg3-7094cdeg3+3325d2eg3-1\
      5881be2g3-2386ce2g3+12516de2g3-1477e3g3+4220bcfg3+13076c2fg3+8871bdfg3-11\
      156cdfg3+2961d2fg3+3286befg3-9095cefg3+2228defg3+15043e2fg3+8109bf2g3+525\
      cf2g3+6607df2g3+15658ef2g3-4313f3g3-9295bcg4+5767c2g4-1418bdg4-2526cdg4-9\
      468d2g4+12357beg4+9982ceg4+11310deg4+10099e2g4-2407bfg4+10345cfg4+11007df\
      g4-5001efg4+15481f2g4+3884bg5+44cg5-11891dg5+12893eg5-3874fg5-11349g6
  ↦ j[75]=ce3f2-4285cde2fg-9052d2e2fg+5254be3fg-8283ce3fg-5905de3fg-9107e4fg-\
      5927c3f2g-13770bcdf2g+2743c2df2g-6878bd2f2g-4291cd2f2g+6717d3f2g-14013bce\
      f2g-10079c2ef2g-12401bdef2g+14496cdef2g-5315d2ef2g+11179be2f2g-1447ce2f2g\
      +6851de2f2g-13408e3f2g+3871bcf3g+11376c2f3g-3024bdf3g-12041cdf3g-13312d2f\
      3g+14762bef3g+14622cef3g+3109def3g+1164e2f3g+3413bf4g+1797cf4g-8483df4g+5\
      178ef4g-4249f5g-7848c4g2+13860bd3g2+5712cd3g2-15303d4g2-9273c3eg2-3326bcd\
      eg2-15963c2deg2-2261bd2eg2+5497cd2eg2-6518d3eg2+11444bce2g2-3369c2e2g2+76\
      02bde2g2+14069cde2g2-13571d2e2g2+4517be3g2+461ce3g2+11581de3g2+6819e4g2+6\
      587c3fg2-10694bcdfg2-10098c2dfg2-11524bd2fg2-14192cd2fg2-1638d3fg2-11542b\
      cefg2-5943c2efg2+2046bdefg2+15400cdefg2+10913d2efg2+9350be2fg2-13810ce2fg\
      2-1991de2fg2+4564e3fg2-2732bcf2g2-1212c2f2g2-819bdf2g2-9288cdf2g2-13973d2\
      f2g2-15015bef2g2-7552cef2g2+344def2g2-10058e2f2g2+1090bf3g2+7883cf3g2-738\
      9df3g2-10401ef3g2-972f4g2+9038c3g3+3505bcdg3+1942c2dg3-8084bd2g3+15202cd2\
      g3-400d3g3+2607bceg3+2439c2eg3-1773bdeg3+1545cdeg3+1235d2eg3-13571be2g3+8\
      74ce2g3+1189de2g3-10058e3g3+15734bcfg3+36c2fg3+14582bdfg3+14752d2fg3+6707\
      befg3+9284cefg3-3915defg3+5075e2fg3-5321bf2g3-676cf2g3-16df2g3-6908ef2g3+\
      11356f3g3+9241bcg4-14905c2g4+13453bdg4+9071cdg4-9384d2g4+5727beg4-15023ce\
      g4-738deg4+13503e2g4+10642bfg4-3718cfg4+2669dfg4+6899efg4-11448f2g4-1835b\
      g5+12590cg5-15879dg5-10988eg5-15258fg5+10265g6
   ↦ j[76]=be3f2-5073cde2fg-7419d2e2fg+12126be3fg+12530ce3fg+15171de3fg+7489e4\
      fg-2093c3f2g-380bcdf2g-4332c2df2g-6023bd2f2g-13278cd2f2g-6472d3f2g+8103bc\
      ef2g-14952c2ef2g+11049bdef2g+9206cdef2g-9364d2ef2g-10519be2f2g-7992ce2f2g\
      -352de2f2g-3246e3f2g-6416bcf3g-7277c2f3g+10757bdf3g-4701cdf3g-5144d2f3g-5\
      242bef3g-9364cef3g-9826def3g-3090e2f3g+11480bf4g-8599cf4g-4734df4g+554ef4\
      g+11429f5g-8288c4g2+2113bd3g2+5756cd3g2-10108d4g2-8883c3eg2+8119bcdeg2-76\
      16c2deg2+10972bd2eg2-12869cd2eg2-15557d3eg2+5356bce2g2+190c2e2g2+12794bde\
      2g2-447cde2g2-10164d2e2g2+5869be3g2-5370ce3g2-6430de3g2-10586e4g2+10155c3\
      fg2+3917bcdfg2-6708c2dfg2-6457bd2fg2-11940cd2fg2+242d3fg2-13922bcefg2-630\
      3c2efg2-6910bdefg2+10972cdefg2-10144d2efg2+14254be2fg2+13754ce2fg2-11859d\
      e2fg2+12074e3fg2-8045bcf2g2-13726c2f2g2+5022bdf2g2+5810cdf2g2-1316d2f2g2+\
      3169bef2g2+13271cef2g2-9639def2g2-7427e2f2g2+5355bf3g2+11615cf3g2-10318df\
      3g2+8303ef3g2-14794f4g2+6445c3g3-15076bcdg3+13550c2dg3-12304bd2g3+3646cd2\
      g3+8400d3g3+11244bceg3+13674c2eg3-14976bdeg3-6851cdeg3+5480d2eg3-13884be2\
      g3+14091ce2g3+15243de2g3+8223e3g3-5821bcfg3+4124c2fg3-14646bdfg3-4908cdfg\
      3-3467d2fg3-11009befg3-13250cefg3+5862defg3+7174e2fg3-12646bf2g3+3752cf2g\
      3-4240df2g3-11346ef2g3-4160f3g3-13201bcg4-508c2g4-13173bdg4+7926cdg4+1411\
      8d2g4-7520beg4-2656ceg4-14743deg4-2337e2g4-6394bfg4-8532cfg4-13713dfg4+12\
      029efg4+7285f2g4-1142bg5+8221cg5+6506dg5-5210eg5+4830fg5+8612g6
   ↦ j[77]=d2e2f2-10785cde2fg+3833d2e2fg+14322be3fg-4658ce3fg+7162de3fg+10799e\
      4fg+12745c3f2g-226bcdf2g+14137c2df2g+11873bd2f2g+4021cd2f2g+15150d3f2g-97\
      24bcef2g-4652c2ef2g+1448bdef2g-5134cdef2g-5148d2ef2g-14890be2f2g+15154ce2\
      f2g-6733de2f2g+8831e3f2g+13137bcf3g+7422c2f3g-14317bdf3g-15416cdf3g-2144d\
      2f3g+7744bef3g+3753cef3g-6852def3g+11114e2f3g+9624bf4g-9206cf4g+2337df4g+\
      10445ef4g+9666f5g+4583c4g2-9650bd3g2-9805cd3g2+6273d4g2+10208c3eg2+12640b\
      cdeg2-3732c2deg2+10938bd2eg2-3125cd2eg2-14071d3eg2+1182bce2g2-5328c2e2g2+\
      13730bde2g2+5752cde2g2+1297d2e2g2-12750be3g2-1377ce3g2+8505de3g2-13167e4g\
      2-4037c3fg2+6141bcdfg2-3679c2dfg2+12306bd2fg2-666cd2fg2+12090d3fg2-4708bc\
      efg2+273c2efg2+6646bdefg2-4094cdefg2-15520d2efg2-15613be2fg2+14364ce2fg2+\
      7913de2fg2+567e3fg2+6093bcf2g2+12960c2f2g2-3198bdf2g2+703cdf2g2+1722d2f2g\
      2+11834bef2g2-13887cef2g2-10666def2g2-15799e2f2g2+945bf3g2-9487cf3g2-418d\
      f3g2+8390ef3g2+4076f4g2-14257c3g3-7707bcdg3-11515c2dg3+14815bd2g3+4295cd2\
      g3-12958d3g3-674bceg3-629c2eg3-10864bdeg3-8296cdeg3-7755d2eg3-10757be2g3+\
      6257ce2g3+7180de2g3-13490e3g3+653bcfg3-7655c2fg3+2845bdfg3-2520cdfg3+1682\
      d2fg3+7319befg3+11175cefg3+9887defg3-13275e2fg3-15279bf2g3-9557cf2g3+7798\
      df2g3-8977ef2g3-2100f3g3-13907bcg4-3720c2g4-1494bdg4-1675cdg4-4145d2g4-80\
      42beg4+13145ceg4-905deg4-12281e2g4-7041bfg4+14726cfg4-10176dfg4+10689efg4\
      +14183f2g4-5394bg5+14046cg5-13605dg5-3223eg5-6389fg5-14623g6
   ↦ j[78]=cde2f2-3622cde2fg+7261d2e2fg-3071be3fg+4776ce3fg-736de3fg-14420e4fg\
      +12640c3f2g-2378bcdf2g-14839c2df2g-12276bd2f2g+629cd2f2g+5687d3f2g+2773bc\
      ef2g+13039c2ef2g+4680bdef2g+11813cdef2g-1005d2ef2g-5923be2f2g+869ce2f2g-1\
      1465de2f2g+10427e3f2g-11483bcf3g-3505c2f3g-3382bdf3g-9149cdf3g-14532d2f3g\
      -460bef3g+5284cef3g+8390def3g+8110e2f3g+11420bf4g+10540cf4g-12264df4g-815\
      6ef4g+13591f5g-7502c4g2-11464bd3g2+883cd3g2+5114d4g2-11575c3eg2+13808bcde\
      g2-2474c2deg2-2664bd2eg2+2703cd2eg2-3015d3eg2-9649bce2g2-14062c2e2g2-6317\
      bde2g2-12418cde2g2+15487d2e2g2-970be3g2+9120ce3g2-4783de3g2-8267e4g2+9324\
      c3fg2-13064bcdfg2+7330c2dfg2-4033bd2fg2+12751cd2fg2+2622d3fg2+6691bcefg2-\
      6585c2efg2+2473bdefg2-9743cdefg2+14008d2efg2+13062be2fg2+6367ce2fg2+5505d\
      e2fg2+3312e3fg2-11262bcf2g2-13209c2f2g2+5993bdf2g2+10905cdf2g2+14283d2f2g\
      2+6455bef2g2-9374cef2g2+4784def2g2-2710e2f2g2-13935bf3g2+8719cf3g2+9326df\
        3g2+13290ef3g2+1552f4g2+12101c3g3+8395bcdg3+9945c2dg3+4602bd2g3+5216cd2g3\
        +9904d3g3+8457bceg3-6602c2eg3-14948bdeg3+12583cdeg3-2303d2eg3+10040be2g3+\
        3912ce2g3-10266de2g3-8995e3g3-13916bcfg3-14361c2fg3+4566bdfg3-12311cdfg3+\
        4618d2fg3+10921befg3+11942cefg3+8992defg3+13452e2fg3+10987bf2g3-2344cf2g3\
        -5398df2g3+7389ef2g3-9443f3g3+15983bcg4-8349c2g4+1333bdg4-500cdg4+6673d2g\
        4+668beg4-11738ceg4+8699deg4+15477e2g4-13069bfg4-8679cfg4-4939dfg4+9667ef\
        g4-4961f2g4+10957bg5-8181cg5-14828dg5-3488eg5-10772fg5+862g6
   ↦ j[79]=cde2fg2+8769d2e2fg2+4585be3fg2-8871ce3fg2+2706de3fg2+13181e4fg2+338\
        4c3f2g2+9062bcdf2g2-10209c2df2g2-6476bd2f2g2-4309cd2f2g2-6652d3f2g2-8928b\
        cef2g2+1062c2ef2g2-14391bdef2g2-10646cdef2g2+9150d2ef2g2-2728be2f2g2-1049\
        8ce2f2g2-13031de2f2g2+11295e3f2g2+10851bcf3g2+8357c2f3g2-4396bdf3g2-10811\
        cdf3g2-107d2f3g2+12131bef3g2-8125cef3g2+12758def3g2-11864e2f3g2+10894bf4g\
        2+2944cf4g2+9590df4g2-6822ef4g2-8625f5g2-8701c4g3-15677bd3g3+12978cd3g3+8\
        989d4g3+15863c3eg3+6375bcdeg3-1588c2deg3-317bd2eg3-1987cd2eg3+7212d3eg3+1\
        0725bce2g3-8331c2e2g3+10408bde2g3+5134cde2g3+12175d2e2g3-4407be3g3+1033ce\
        3g3+767de3g3+12392e4g3-9473c3fg3-14788bcdfg3-14687c2dfg3+9552bd2fg3-1265c\
        d2fg3+6327d3fg3+7567bcefg3-4304c2efg3+1545bdefg3-15201cdefg3+13499d2efg3-\
        8803be2fg3+8604ce2fg3-5072de2fg3-4039e3fg3+6944bcf2g3-663c2f2g3-12344bdf2\
        g3-8884cdf2g3+11132d2f2g3+1833bef2g3+15224cef2g3-13616def2g3-12875e2f2g3+\
        10400bf3g3+5460cf3g3+6537df3g3-4935ef3g3+4616f4g3+6158c3g4-11397bcdg4-243\
        1c2dg4+6039bd2g4-15696cd2g4-15174d3g4-11618bceg4+1778c2eg4-936bdeg4+14771\
        cdeg4+7638d2eg4+12781be2g4+794ce2g4-9775de2g4+1666e3g4-9686bcfg4-11383c2f\
        g4+12633bdfg4+3211cdfg4+474d2fg4-11956befg4-7635cefg4-14672defg4-7000e2fg\
        4-13439bf2g4+2713cf2g4+4322df2g4+13516ef2g4+9546f3g4-11844bcg5+7386c2g5+7\
        895bdg5+313cdg5+14055d2g5+14138beg5+4035ceg5+2103deg5+630e2g5+1436bfg5+13\
        727cfg5-13263dfg5-10836efg5+5487f2g5-3663bg6+7819cg6+8625dg6+14802eg6+996\
        8fg6-10115g7-10502
   ↦ j[80]=e4fg3+7972c3f2g3+1271bcdf2g3+9000c2df2g3+5645bd2f2g3+2760cd2f2g3-39\
        05d3f2g3-9191bcef2g3-7947c2ef2g3-11090bdef2g3-15959cdef2g3-5047d2ef2g3-54\
        14be2f2g3-5854ce2f2g3+8544de2f2g3+3316e3f2g3+14107bcf3g3-2596c2f3g3-3540b\
        df3g3+505cdf3g3-480d2f3g3+8452bef3g3-9950cef3g3-9906def3g3+7640e2f3g3-903\
        0bf4g3-7667cf4g3+12855df4g3+12840ef4g3+5270f5g3-12208c4g4+2383bd3g4+11610\
        cd3g4-11958d4g4-6259c3eg4-10315bcdeg4+7584c2deg4+1642bd2eg4-3963cd2eg4+11\
        239d3eg4-15208bce2g4-11944c2e2g4+9920bde2g4+2355cde2g4+2336d2e2g4-11556be\
        3g4+7156ce3g4-4494de3g4-8246e4g4+291c3fg4+7669bcdfg4-9408c2dfg4+8412bd2fg\
        4+7071cd2fg4-5298d3fg4-5656bcefg4-4212c2efg4+6375bdefg4+3814cdefg4+3646d2\
        efg4-5211be2fg4+3663ce2fg4+9356de2fg4-8961e3fg4-7682bcf2g4-15418c2f2g4-14\
        013bdf2g4-15242cdf2g4-14349d2f2g4-15359bef2g4+9663cef2g4-14429def2g4-6050\
        e2f2g4+11607bf3g4+6464cf3g4+11244df3g4-6920ef3g4-2327f4g4-11259c3g5-12673\
        bcdg5+11008c2dg5+8116bd2g5+3431cd2g5+8730d3g5-7403bceg5+14970c2eg5+12575b\
        deg5+2705cdeg5-7732d2eg5-13526be2g5-9434ce2g5-8363de2g5-9935e3g5-6410bcfg\
        5-7729c2fg5-8988bdfg5-13237cdfg5+6851d2fg5-949befg5-5172cefg5-6158defg5+1\
        4751e2fg5-10176bf2g5-12123cf2g5-12773df2g5-11525ef2g5+517f3g5+14702bcg6-4\
        975c2g6-3481bdg6+5981cdg6+2740d2g6+3768beg6-7571ceg6+6008deg6-11949e2g6-4\
        029bfg6+2110cfg6+10249dfg6+5025efg6+1383f2g6-4568bg7+14137cg7-6515dg7-472\
        2eg7+2023fg7-2782g8+14818b-14485c+4985d-5652e+12008f+11272g
   ↦ j[81]=de3fg3-15436c3f2g3+13103bcdf2g3+13134c2df2g3-5586bd2f2g3-712cd2f2g3\
        -15260d3f2g3-4500bcef2g3-15191c2ef2g3-5680bdef2g3-2700cdef2g3+2812d2ef2g3\
        +2821be2f2g3-2870ce2f2g3+7439de2f2g3+10137e3f2g3-2238bcf3g3+13339c2f3g3+4\
        353bdf3g3-15664cdf3g3+15191d2f3g3-13103bef3g3-12444cef3g3+11318def3g3+111\
        0e2f3g3-3123bf4g3+13864cf4g3+10606df4g3+4987ef4g3+4847f5g3-5417c4g4+10212\
        bd3g4+5088cd3g4+9307d4g4-856c3eg4+13266bcdeg4+471c2deg4-5324bd2eg4+6610cd\
        2eg4-1453d3eg4+9697bce2g4-5611c2e2g4+9718bde2g4-14113cde2g4-5587d2e2g4+10\
        700be3g4+3631ce3g4+2572de3g4-11941e4g4-10436c3fg4+7085bcdfg4-4650c2dfg4+1\
        0627bd2fg4-10293cd2fg4-9553d3fg4-6463bcefg4-15001c2efg4+6675bdefg4+11555c\
        defg4+12235d2efg4-3214be2fg4+297ce2fg4-10247de2fg4+12620e3fg4-8725bcf2g4+\
        6856c2f2g4-12748bdf2g4+8336cdf2g4+13809d2f2g4-3158bef2g4+11888cef2g4+7102\
        def2g4+11730e2f2g4-2320bf3g4+13657cf3g4-5740df3g4-6942ef3g4-2998f4g4+6949\
        c3g5+15296bcdg5+14479c2dg5+6767bd2g5-4095cd2g5-15636d3g5+3914bceg5-14471c\
        2eg5+4763bdeg5-2157cdeg5+13154d2eg5+3489be2g5+11935ce2g5-9877de2g5-11564e\
        3g5-3883bcfg5-7753c2fg5-343bdfg5-3017cdfg5-3372d2fg5+1601befg5+4422cefg5+\
        6484defg5+826e2fg5+10796bf2g5-1553cf2g5+4095df2g5-4078ef2g5+8829f3g5-1260\
        4bcg6+3564c2g6-7834bdg6-13260cdg6+12544d2g6+2938beg6-8964ceg6+10142deg6-1\
        0306e2g6+14450bfg6-1508cfg6-9835dfg6+14497efg6-13969f2g6-7693bg7-7442cg7+\
        8310dg7-1901eg7-11444fg7-13605g8+10274b+7153c-8069d-3057e-12464f-11423g
     ↦ j[82]=ce3fg3+5524c3f2g3+786bcdf2g3+2289c2df2g3-9687bd2f2g3-13816cd2f2g3-1\
        105d3f2g3+4643bcef2g3-5128c2ef2g3-6677bdef2g3-8039cdef2g3+15601d2ef2g3-17\
        97be2f2g3-351ce2f2g3+9157de2f2g3-314e3f2g3+14644bcf3g3+10618c2f3g3+6508bd\
        f3g3+9309cdf3g3+10684d2f3g3-14681bef3g3+3833cef3g3-5043def3g3+14504e2f3g3\
        -10006bf4g3+8350cf4g3-3671df4g3-12750ef4g3+4975f5g3-247c4g4-7023bd3g4+185\
        4cd3g4-10855d4g4+13649c3eg4-14842bcdeg4+12031c2deg4+12400bd2eg4-4590cd2eg\
        4-4704d3eg4+14918bce2g4-6270c2e2g4-8198bde2g4+13798cde2g4-11962d2e2g4-869\
        1be3g4-5318ce3g4+15395de3g4+11018e4g4+11588c3fg4-12864bcdfg4+2792c2dfg4+8\
        939bd2fg4+15801cd2fg4+14931d3fg4-13975bcefg4-12407c2efg4-3898bdefg4+6085c\
        defg4-11896d2efg4+12665be2fg4-167ce2fg4+11855de2fg4+9520e3fg4+11399bcf2g4\
        +11552c2f2g4-1673bdf2g4-3384cdf2g4+5857d2f2g4+6784bef2g4+10275cef2g4-6777\
        def2g4-15063e2f2g4+8011bf3g4+13795cf3g4+2182df3g4-9264ef3g4-2024f4g4-1040\
        0c3g5+14352bcdg5-4530c2dg5+10888bd2g5+307cd2g5+2511d3g5-1037bceg5+3034c2e\
        g5+15350bdeg5-10488cdeg5+14956d2eg5+1172be2g5+14059ce2g5-166de2g5+13617e3\
        g5-14258bcfg5-929c2fg5-2426bdfg5-13911cdfg5+11647d2fg5+13948befg5-15328ce\
        fg5+10386defg5+11780e2fg5+47bf2g5-13518cf2g5-6648df2g5-1712ef2g5-821f3g5-\
        12728bcg6+364c2g6+6951bdg6+15200cdg6-10143d2g6+15175beg6+11498ceg6+6351de\
        g6-13137e2g6+8622bfg6+11723cfg6+2028dfg6+8163efg6-15223f2g6+4665bg7-2867c\
        g7-4022dg7-8598eg7+514fg7+5780g8+15600b-5093c-11908d-5447e+9119f+6186g
     ↦ j[83]=be3fg3-13505c3f2g3-14373bcdf2g3+6680c2df2g3+3220bd2f2g3-8219cd2f2g3\
        -14673d3f2g3-15710bcef2g3+1829c2ef2g3-14672bdef2g3-33cdef2g3-3011d2ef2g3+\
        14382be2f2g3+8ce2f2g3-12637de2f2g3-13391e3f2g3+13417bcf3g3-12186c2f3g3+31\
        58bdf3g3-6199cdf3g3-11542d2f3g3-9548bef3g3-8238cef3g3-13204def3g3+2606e2f\
        3g3+8012bf4g3-9230cf4g3-5751df4g3-12693ef4g3+5503f5g3-3995c4g4+11361bd3g4\
        -13757cd3g4-15622d4g4+4946c3eg4-10607bcdeg4+7288c2deg4+786bd2eg4-10277cd2\
        eg4+146d3eg4-107bce2g4-13122c2e2g4-1626bde2g4-9101cde2g4+14659d2e2g4-947b\
        e3g4-6112ce3g4+1604de3g4-9232e4g4-15539c3fg4-11906bcdfg4+8051c2dfg4-12184\
        bd2fg4-8112cd2fg4-455d3fg4-13257bcefg4+9925c2efg4-10877bdefg4+6704cdefg4-\
        14103d2efg4+10728be2fg4+428ce2fg4-14871de2fg4-2283e3fg4+9933bcf2g4-4901c2\
        f2g4+14183bdf2g4+13307cdf2g4+15499d2f2g4-12659bef2g4-13436cef2g4+10437def\
        2g4+2568e2f2g4-5152bf3g4-15925cf3g4-14547df3g4-11860ef3g4+9174f4g4+10912c\
        3g5-4693bcdg5-2771c2dg5-6182bd2g5+15497cd2g5+7858d3g5-13739bceg5-13078c2e\
        g5+7292bdeg5+1480cdeg5+7074d2eg5-6665be2g5+8821ce2g5-13923de2g5+1817e3g5+\
        511bcfg5+6909c2fg5-5680bdfg5+4217cdfg5+12187d2fg5+12832befg5+15294cefg5+1\
        3715defg5-55e2fg5+8783bf2g5-13553cf2g5-8500df2g5+10551ef2g5+9439f3g5+1890\
        bcg6+5894c2g6+6336bdg6+9907cdg6-3046d2g6+65beg6+7648ceg6-13110deg6-1085e2\
        g6-6870bfg6+4917cfg6+6197dfg6+11105efg6+1631f2g6-15658bg7-7594cg7+12309dg\
        7+11304eg7-14652fg7+2788g8+2216b-8169c+4875d-10907e+492f-2007g
     ↦ j[84]=d2e2fg3-4740c3f2g3-8816bcdf2g3-6397c2df2g3-3820bd2f2g3+5091cd2f2g3+\
        4473d3f2g3+8675bcef2g3+6460c2ef2g3+3896bdef2g3-14160cdef2g3+11807d2ef2g3+\
        6033be2f2g3-11342ce2f2g3-11393de2f2g3+6272e3f2g3+8677bcf3g3+13804c2f3g3+7\
        56bdf3g3+14217cdf3g3-7266d2f3g3+6828bef3g3+551cef3g3+4647def3g3-14506e2f3\
        g3+13454bf4g3-7317cf4g3-13083df4g3-324ef4g3+2481f5g3+12786c4g4-1331bd3g4-\
        1936cd3g4-2086d4g4+11651c3eg4-12651bcdeg4+7355c2deg4+14705bd2eg4+8439cd2e\
        g4+8089d3eg4-14007bce2g4+8220c2e2g4-3652bde2g4-14752cde2g4+9700d2e2g4-145\
        17be3g4-6247ce3g4-7405de3g4+6005e4g4+6080c3fg4-2937bcdfg4-15819c2dfg4+146\
        86bd2fg4-11500cd2fg4-13994d3fg4+8084bcefg4-14117c2efg4+11643bdefg4-10856c\
        defg4-213d2efg4+14704be2fg4+10746ce2fg4+12279de2fg4-3358e3fg4-10787bcf2g4\
        -10675c2f2g4+14039bdf2g4-11322cdf2g4+10095d2f2g4-4469bef2g4-6922cef2g4+13\
        341def2g4-10449e2f2g4-14520bf3g4+8928cf3g4+9254df3g4+3797ef3g4-2501f4g4-4\
        683c3g5+12937bcdg5+4155c2dg5-618bd2g5+521cd2g5-8847d3g5+13642bceg5+10999c\
        2eg5+5360bdeg5+1492cdeg5-12290d2eg5-11427be2g5-5335ce2g5-4454de2g5-3104e3\
        g5+471bcfg5+14664c2fg5-5320bdfg5-9097cdfg5-11983d2fg5+1705befg5+288cefg5+\
        13114defg5-11150e2fg5+4503bf2g5-12977cf2g5-13631df2g5+3662ef2g5-4695f3g5+\
        9352bcg6-4476c2g6+1760bdg6-7908cdg6+14070d2g6+489beg6+9908ceg6-5351deg6-1\
        332e2g6+4793bfg6+13921cfg6+7363dfg6+6456efg6+15704f2g6+7274bg7+11177cg7+1\
        4357dg7-12791eg7+15025fg7-904g8-15588b-514c+12356d+1704e+2169f+12531g
     ↦ j[85]=de2f2g4-11965e3f2g4-6571bcf3g4-11437c2f3g4-10159bdf3g4-3273cdf3g4+4\
        295d2f3g4+14252bef3g4+1254cef3g4+11711def3g4+1493e2f3g4+8761bf4g4+12621cf\
        4g4+11735df4g4-3533ef4g4+5084f5g4-2530c4g5+4213bd3g5-3638cd3g5-12794d4g5+\
        4582c3eg5+6054bcdeg5-140c2deg5+5072bd2eg5-3793cd2eg5-11433d3eg5-12336bce2\
        g5-270c2e2g5-8546bde2g5+9133cde2g5-646d2e2g5+5007be3g5-11031ce3g5+9652de3\
        g5+15086e4g5+4240c3fg5-1076bcdfg5+12192c2dfg5+9373bd2fg5-11824cd2fg5+2230\
        d3fg5+6322bcefg5+1767c2efg5+15107bdefg5-13673cdefg5-6437d2efg5-3127be2fg5\
        -11555ce2fg5-3560de2fg5+2450e3fg5+8814bcf2g5-4128c2f2g5+301bdf2g5+10117cd\
        f2g5-4734d2f2g5-6674bef2g5+1791cef2g5+14489def2g5-866e2f2g5+2921bf3g5-385\
        1cf3g5+1466df3g5+4395ef3g5-6728f4g5-15158c3g6-770bcdg6-7232c2dg6+10496bd2\
        g6-8432cd2g6+7372d3g6-3645bceg6-13437c2eg6-10710bdeg6+13101cdeg6-10189d2e\
        g6+1687be2g6-4854ce2g6-4674de2g6-6791e3g6+5972bcfg6+4464c2fg6-2096bdfg6+3\
        516cdfg6+7128d2fg6-14947befg6-4523cefg6+2813defg6+10490e2fg6+5393bf2g6+12\
        829cf2g6+3488df2g6+11366ef2g6+1654f3g6-12197bcg7+8383c2g7-788bdg7-13956cd\
        g7+14402d2g7+11974beg7-9580ceg7-14671deg7-15604e2g7-7877bfg7+12264cfg7-69\
        48dfg7-11692efg7-3006f2g7+6111bg8-8848cg8+10314dg8+2521eg8+2938fg8+9550g9\
        -9515bc-15857c2+14931bd+6522cd-879d2+4652be-9025ce-3420de-7315e2+14919bf+\
        15569cf+8733df-13238ef+370f2+12708bg-1253cg-5342dg-9585eg-7004fg+775g2
     ↦ j[86]=ce2f2g4+15814e3f2g4-392bcf3g4-857c2f3g4+13445bdf3g4+1530cdf3g4+1378\
        0d2f3g4-1119bef3g4+3357cef3g4+12200def3g4+1336e2f3g4+11860bf4g4+13981cf4g\
        4-15692df4g4+9903ef4g4+1807f5g4-14522c4g5+3420bd3g5+11613cd3g5+5533d4g5-1\
        4871c3eg5-540bcdeg5+7210c2deg5+11915bd2eg5-15217cd2eg5-11366d3eg5-10832bc\
        e2g5-15716c2e2g5+12905bde2g5-12303cde2g5-14130d2e2g5-15310be3g5+15547ce3g\
        5-85de3g5-15968e4g5+7155c3fg5-10980bcdfg5+7079c2dfg5-10958bd2fg5+12672cd2\
        fg5+1898d3fg5+7343bcefg5-1424c2efg5+7893bdefg5-8795cdefg5-2130d2efg5-1362\
        7be2fg5-449ce2fg5-6313de2fg5+5246e3fg5-15922bcf2g5+12316c2f2g5-13768bdf2g\
        5-13260cdf2g5-8376d2f2g5+5457bef2g5-12651cef2g5+7618def2g5+12366e2f2g5+40\
        23bf3g5+1423cf3g5-12407df3g5-3266ef3g5+10641f4g5+3347c3g6+1023bcdg6-13684\
        c2dg6-2349bd2g6-13303cd2g6+2749d3g6+4317bceg6-8373c2eg6+2377bdeg6+9270cde\
        g6-1485d2eg6-3843be2g6-9736ce2g6-6599de2g6-13745e3g6-876bcfg6-4318c2fg6-1\
        0889bdfg6-11865cdfg6+9367d2fg6-1941befg6-10073cefg6-2184defg6-7411e2fg6-5\
        323bf2g6+14286cf2g6-1764df2g6-1145ef2g6+10676f3g6-12662bcg7+3299c2g7+8166\
        bdg7+4384cdg7+12281d2g7+15629beg7+961ceg7-2412deg7-10653e2g7+3111bfg7+159\
        25cfg7+3763dfg7+14857efg7+12084f2g7+7022bg8+11284cg8+11251dg8-11879eg8+56\
        8fg8-14325g9-6580bc-14693c2-8236bd-5213cd+8723d2-4314be+11583ce+14819de-9\
        06e2+3047bf-15172cf-9597df+8928ef-14526f2-4495bg-502cg-5390dg+4583eg+1210\
        fg-3476g2
```

```
↦ j[87]=be2f2g4-3146e3f2g4+7057bcf3g4+15530c2f3g4+1604bdf3g4+14540cdf3g4+12\
  454d2f3g4-12607bef3g4+10482cef3g4+88def3g4+15727e2f3g4+3612bf4g4-1873cf4g\
  4+2649df4g4-10634ef4g4+3118f5g4+15765c4g5-14637bd3g5+612cd3g5+9822d4g5+11\
  147c3eg5-6972bcdeg5+9961c2deg5-3577bd2eg5+8810cd2eg5-14415d3eg5-14831bce2\
  g5+10862c2e2g5+7299bde2g5-3060cde2g5-10442d2e2g5+4695be3g5+531ce3g5+3475d\
  e3g5+7122e4g5-11995c3fg5+10479bcdfg5-8694c2dfg5+12496bd2fg5+1695cd2fg5+14\
  22d3fg5-10473bcefg5+4193c2efg5+3863bdefg5+9052cdefg5-8861d2efg5+14140be2f\
  g5-10386ce2fg5+11879de2fg5+11159e3fg5+7412bcf2g5+681c2f2g5-6321bdf2g5-915\
  9cdf2g5+15306d2f2g5-5653bef2g5+13351cef2g5+135def2g5-15405e2f2g5+12679bf3\
  g5+6310cf3g5+2203df3g5+9721ef3g5+7493f4g5-15953c3g6+11280bcdg6-5064c2dg6-\
  11894bd2g6+13093cd2g6-15731d3g6+11341bceg6+4507c2eg6-15017bdeg6-767cdeg6-\
  13014d2eg6-2136be2g6+7624ce2g6+11387de2g6-198e3g6-8478bcfg6+2667c2fg6+597\
  8bdfg6+7073cdfg6-14689d2fg6+7019befg6-322cefg6-14294defg6+13337e2fg6-1280\
  6bf2g6-10649cf2g6+4436df2g6-14805ef2g6+7653f3g6-4127bcg7-5910c2g7-13206bd\
  g7+8994cdg7-6636d2g7-6840beg7+1450ceg7-1802deg7+10914e2g7-12118bfg7-9759c\
  fg7+8624dfg7-10949efg7+10670f2g7-3494bg8-10039cg8+8045dg8+505eg8-12273fg8\
  +8397g9-3334bc+4156c2+7377bd-2564cd-2878d2-14781be-8192ce+50de-1925e2-930\
  7bf+15622cf-8765df-7079ef-9597f2+271bg-7398cg-2771dg+4143eg-12249fg-2294g\
  2
↦ j[88]=d2ef2g4+15618e3f2g4+8689bcf3g4-3945c2f3g4+13923bdf3g4+2575cdf3g4+12\
  729d2f3g4-1696bef3g4-6365cef3g4-429def3g4-6259e2f3g4+3038bf4g4-2524cf4g4-\
  11099df4g4+5915ef4g4-9139f5g4+5649c4g5+8296bd3g5-3517cd3g5-14508d4g5-9055\
  c3eg5+11219bcdeg5-15536c2deg5+5640bd2eg5+14226cd2eg5+9414d3eg5+2190bce2g5\
  +3497c2e2g5+5650bde2g5+12776cde2g5+8965d2e2g5+14648be3g5+7311ce3g5-9779de\
  3g5-10458e4g5-10287c3fg5-15711bcdfg5+12163c2dfg5+9596bd2fg5+10074cd2fg5+1\
  241d3fg5+15464bcefg5+14635c2efg5+7888bdefg5+7379cdefg5-13087d2efg5+817be2\
  fg5-7554ce2fg5-15937de2fg5-15833e3fg5+10289bcf2g5+6209c2f2g5-14694bdf2g5+\
  3848cdf2g5+8679d2f2g5+3622bef2g5+4081cef2g5-9784def2g5+13882e2f2g5+14142b\
  f3g5-9131cf3g5+1634df3g5-6962ef3g5-1555f4g5+6808c3g6-12611bcdg6+2119c2dg6\
  -5411bd2g6+7196cd2g6+14382d3g6+5135bceg6-14560c2eg6-6014bdeg6-11201cdeg6+\
  13589d2eg6+1452be2g6-7509ce2g6-8059de2g6-12560e3g6-4111bcfg6-3420c2fg6+97\
  53bdfg6-5429cdfg6+14292d2fg6-4971befg6-10081cefg6-6047defg6-12741e2fg6+23\
  2bf2g6-7256cf2g6+9358df2g6+15315ef2g6-9708f3g6-7326bcg7-4488c2g7-5107bdg7\
  +4040cdg7-14485d2g7-6516beg7-10658ceg7+13756deg7+10306e2g7-15219bfg7-1173\
  2cfg7+14210dfg7-8599efg7+223f2g7-11453bg8+2308cg8-553dg8-2504eg8+15324fg8\
  -5197g9+5372bc-13179c2-508bd-1204cd+13796d2+12589be-4722ce-10035de+2932e2\
  +8935bf-6801cf+9838df+11150ef+7097f2-3225bg+15116cg+7091dg-5929eg-894fg+9\
  547g2
↦ j[89]=cdef2g4+8633e3f2g4-15395bcf3g4+2088c2f3g4-7495bdf3g4-10035cdf3g4-28\
  82d2f3g4+2859bef3g4+15084cef3g4+7917def3g4-11741e2f3g4+4885bf4g4+3981cf4g\
  4+6028df4g4+7795ef4g4+501f5g4+8168c4g5+5650bd3g5+8320cd3g5-5698d4g5-8735c\
  3eg5+12896bcdeg5-7153c2deg5-10240bd2eg5+11810cd2eg5+8889d3eg5+14204bce2g5\
  -5079c2e2g5-732bde2g5+15741cde2g5+2062d2e2g5+15464be3g5+10636ce3g5-4978de\
  3g5+12266e4g5+1517c3fg5+7302bcdfg5-6411c2dfg5+4009bd2fg5+197cd2fg5-4595d3\
  fg5-7519bcefg5-12179c2efg5-15484bdefg5-9077cdefg5+12470d2efg5-10963be2fg5\
  +9360ce2fg5+11070de2fg5+6508e3fg5+15971bcf2g5+9874c2f2g5+8667bdf2g5+2579c\
  df2g5+4377d2f2g5-1625bef2g5+7485cef2g5+11369def2g5-4262e2f2g5+10756bf3g5+\
  4138cf3g5+6829df3g5+2517ef3g5-523f4g5-4475c3g6-8030bcdg6+4046c2dg6-2880bd\
  2g6-8787cd2g6+11037d3g6-3836bceg6-5719c2eg6-6036bdeg6-10212cdeg6-12459d2e\
  g6+13255be2g6-9325ce2g6+3725de2g6-8855e3g6+6772bcfg6-918c2fg6-14458bdfg6-\
  9568cdfg6+15673d2fg6-7541befg6-3852cefg6-14078defg6+5293e2fg6+14869bf2g6+\
  13085cf2g6-14667df2g6+14807ef2g6+9634f3g6+11132bcg7+13994c2g7+144bdg7-391\
  7cdg7+6751d2g7+11610beg7+4392ceg7+6830deg7-7843e2g7+6506bfg7+11457cfg7+45\
   21dfg7-10325efg7-15414f2g7-12674bg8+10458cg8+12230dg8-13808eg8+7576fg8-33\
   0g9-8160bc+14680c2-14476bd+8103cd+10241d2+7192be+2401ce-11203de+13098e2-1\
   2752bf-693cf+4105df+7790ef+13499f2-13847bg+4860cg-4987dg+2989eg+14496fg-1\
   5180g2
↦ j[90]=bdef2g4-8790e3f2g4+258bcf3g4-10989c2f3g4+6662bdf3g4+2512cdf3g4-7965\
   d2f3g4-7074bef3g4-11586cef3g4-2512def3g4+13525e2f3g4+5768bf4g4-8800cf4g4+\
   7404df4g4+3758ef4g4+4433f5g4-4178c4g5+13328bd3g5-15742cd3g5-12201d4g5-179\
   9c3eg5-3527bcdeg5-4360c2deg5-11391bd2eg5+478cd2eg5-100d3eg5-3497bce2g5+58\
   38c2e2g5-3249bde2g5-3795cde2g5-14178d2e2g5-11334be3g5+1214ce3g5-7835de3g5\
   +6027e4g5-9264c3fg5+15666bcdfg5-4793c2dfg5-1066bd2fg5+5371cd2fg5+15280d3f\
   g5+13036bcefg5+1088c2efg5+7776bdefg5+12683cdefg5+6870d2efg5-3463be2fg5+51\
   40ce2fg5-9180de2fg5+15505e3fg5+10025bcf2g5+416c2f2g5-12121bdf2g5+7566cdf2\
   g5+6767d2f2g5+13293bef2g5-11488cef2g5-13039def2g5-9516e2f2g5+2462bf3g5+40\
   94cf3g5-3042df3g5-13968ef3g5-14058f4g5-11205c3g6-15597bcdg6-6601c2dg6+769\
   9bd2g6+4394cd2g6+15400d3g6-3067bceg6-15648c2eg6+2958bdeg6-10709cdeg6-1047\
   5d2eg6+1754be2g6+15077ce2g6-10794de2g6+1475e3g6+6518bcfg6-13983c2fg6-1271\
   4bdfg6+15282cdfg6-2864d2fg6-7974befg6+3379cefg6-934defg6+13429e2fg6-3339b\
   f2g6+6349cf2g6+303df2g6-753ef2g6-1670f3g6+1080bcg7-13079c2g7-5847bdg7-987\
   0cdg7+11490d2g7+8278beg7+5165ceg7+12606deg7-11316e2g7+3477bfg7-15689cfg7+\
   15244dfg7+2483efg7-760f2g7-255bg8-3129cg8-14465dg8+10084eg8+5237fg8+1327g\
   9-1391bc+12097c2+8584bd-5965cd-3528d2-9026be-5697ce-2274de+3072e2-12412bf\
   -1809cf-15779df+11998ef+12849f2-279bg-8019cg+8163dg-8683eg+6612fg-6617g2
↦ j[91]=c2ef2g4-745e3f2g4-5383bcf3g4+12613c2f3g4+13434bdf3g4+891cdf3g4+4748\
   d2f3g4+13487bef3g4-2630cef3g4-10782def3g4+13787e2f3g4+9955bf4g4-6834cf4g4\
   +7806df4g4+903ef4g4+5286f5g4+13109c4g5-14491bd3g5-5545cd3g5-631d4g5+11437\
   c3eg5-4025bcdeg5+8668c2deg5+8020bd2eg5+11071cd2eg5+14241d3eg5+14276bce2g5\
   -3414c2e2g5-11192bde2g5-2348cde2g5-5741d2e2g5+13389be3g5-2795ce3g5+9470de\
   3g5-11329e4g5-11665c3fg5-4886bcdfg5+11325c2dfg5-4064bd2fg5-14727cd2fg5+14\
   704d3fg5+11924bcefg5-7606c2efg5+13749bdefg5-2697cdefg5-5284d2efg5+11882be\
   2fg5-5609ce2fg5+10384de2fg5+4094e3fg5+12901bcf2g5-11483c2f2g5+5613bdf2g5-\
   3435cdf2g5+13121d2f2g5+3679bef2g5+8309cef2g5+4105def2g5-4407e2f2g5-15586b\
   f3g5-8995cf3g5-6874df3g5-993ef3g5-12982f4g5+1661c3g6+15714bcdg6-2786c2dg6\
   -5527bd2g6-9736cd2g6-11026d3g6-5900bceg6+7867c2eg6+5128bdeg6-1417cdeg6-11\
   847d2eg6-9982be2g6+8354ce2g6-7226de2g6-15077e3g6-6330bcfg6+11387c2fg6-107\
   11bdfg6-10290cdfg6+11685d2fg6+13604befg6-1128cefg6+9116defg6+9257e2fg6+15\
   195bf2g6+3420cf2g6+4803df2g6+12373ef2g6-3581f3g6-5319bcg7-870c2g7+5286bdg\
   7-3358cdg7-2364d2g7+12142beg7+2131ceg7+725deg7-1702e2g7-10354bfg7-14032cf\
   g7-12475dfg7-12521efg7+13652f2g7+15677bg8+15628cg8+5895dg8+3036eg8-4361fg\
   8-7560g9-15095bc+13754c2+14084bd-4276cd-7837d2-5777be+9028ce+655de+9046e2\
   +13447bf-4432cf+846df-2439ef+4022f2-11955bg+11839cg-8144dg-2550eg-14252fg\
   +8588g2
↦ j[92]=bcef2g4-6162e3f2g4-11655bcf3g4+11402c2f3g4-10471bdf3g4-6352cdf3g4+3\
   319d2f3g4+1746bef3g4+7240cef3g4+13695def3g4-7198e2f3g4-12767bf4g4-15773cf\
   4g4-5838df4g4+1533ef4g4+9436f5g4+4233c4g5+609bd3g5+2674cd3g5-10810d4g5-36\
   14c3eg5+1965bcdeg5-15061c2deg5-12365bd2eg5-12576cd2eg5-11096d3eg5-10962bc\
   e2g5-8144c2e2g5-13754bde2g5+967cde2g5-3573d2e2g5+14072be3g5+11280ce3g5+94\
   82de3g5+10220e4g5-4913c3fg5+13660bcdfg5-12301c2dfg5+14988bd2fg5+2853cd2fg\
   5+2865d3fg5-8919bcefg5-6168c2efg5-8764bdefg5+1909cdefg5-4118d2efg5-6464be\
   2fg5+5173ce2fg5+13985de2fg5+409e3fg5+1064bcf2g5-9534c2f2g5-6433bdf2g5-116\
   61cdf2g5-13729d2f2g5+14469bef2g5-6531cef2g5+9788def2g5-3319e2f2g5+12182bf\
   3g5+14034cf3g5+5618df3g5+11630ef3g5-12249f4g5-4232c3g6-7977bcdg6+15566c2d\
   g6-8571bd2g6-4467cd2g6+15395d3g6+1417bceg6+3973c2eg6+8315bdeg6+15517cdeg6\
   +2261d2eg6+373be2g6-5544ce2g6-1161de2g6-1311e3g6+6701bcfg6-2791c2fg6-1007\
   0bdfg6+6019cdfg6+6532d2fg6-13928befg6-14621cefg6+15736defg6+11915e2fg6+15\
   647bf2g6+4762cf2g6+12819df2g6+8606ef2g6+11040f3g6-14782bcg7+5345c2g7+1145\
   2bdg7-9108cdg7+9594d2g7+6937beg7-8877ceg7-15534deg7+3754e2g7-3873bfg7+584\
   2cfg7-10053dfg7-14054efg7-465f2g7-11159bg8-11958cg8-14323dg8+2507eg8-5261\
   fg8-9327g9+15335bc-4123c2+14801bd-9066cd-12261d2-11188be+8000ce-13575de+1\
   5648e2+11364bf+14999cf-5201df-3372ef-6678f2-14999bg+9182cg-11260dg+2292eg\
   +13094fg+15200g2
↦ j[93]=d3f2g4+5252e3f2g4+14056bcf3g4-3867c2f3g4-8712bdf3g4-188cdf3g4+12647\
   d2f3g4+9306bef3g4-442cef3g4+10582def3g4-2331e2f3g4-11787bf4g4+11405cf4g4+\
   563df4g4+6588ef4g4-5243f5g4-675c4g5-1543bd3g5+6277cd3g5+2029d4g5+7718c3eg\
   5-14497bcdeg5-14633c2deg5+4775bd2eg5+7156cd2eg5+9214d3eg5-9183bce2g5+1096\
   3c2e2g5+1201bde2g5+15597cde2g5+13326d2e2g5-9792be3g5+3920ce3g5-450de3g5+1\
   1866e4g5-7340c3fg5-15158bcdfg5-3569c2dfg5-9645bd2fg5-411cd2fg5-8340d3fg5+\
   4527bcefg5+13632c2efg5-3747bdefg5-3218cdefg5-6787d2efg5-4765be2fg5+235ce2\
   fg5+1638de2fg5-3722e3fg5+865bcf2g5+13844c2f2g5+5095bdf2g5-10787cdf2g5-791\
   7d2f2g5-14215bef2g5+2298cef2g5+3818def2g5-7645e2f2g5+4328bf3g5+13320cf3g5\
   +8481df3g5-13351ef3g5-7020f4g5+8572c3g6+641bcdg6-1155c2dg6+3505bd2g6-7497\
   cd2g6-4591d3g6-11712bceg6-11801c2eg6-15484bdeg6+8256cdeg6+7813d2eg6+10275\
   be2g6+197ce2g6+2615de2g6+4501e3g6-3595bcfg6+6053c2fg6+15082bdfg6-15716cdf\
   g6-5859d2fg6+7887befg6-174cefg6+2854defg6-5379e2fg6-12276bf2g6-357cf2g6-9\
   566df2g6-4644ef2g6+9587f3g6-1823bcg7-8073c2g7+6159bdg7+13517cdg7+14679d2g\
   7-12338beg7+9343ceg7+8587deg7+5900e2g7+15089bfg7+6507cfg7-3976dfg7-6856ef\
   g7+13889f2g7+4269bg8-11684cg8-11136dg8-14818eg8-4164fg8+12112g9-6831bc+51\
   96c2-9377bd+7094cd+4128d2+15696be+190ce+6405de-9851e2+9414bf-8464cf+2969d\
   f+9637ef-5365f2+12307bg-7115cg+8165dg+12134eg+2629fg+8290g2
↦ j[94]=cd2f2g4-2147e3f2g4+3687bcf3g4+13220c2f3g4+9217bdf3g4-13329cdf3g4+10\
   749d2f3g4-8572bef3g4+15483cef3g4+15278def3g4-9836e2f3g4+4788bf4g4-15226cf\
   4g4+15155df4g4-12621ef4g4-1285f5g4-13709c4g5-5022bd3g5+8277cd3g5+3684d4g5\
   -8935c3eg5-11786bcdeg5+12685c2deg5-1189bd2eg5-7002cd2eg5-2371d3eg5-14243b\
   ce2g5-12394c2e2g5-3962bde2g5+1881cde2g5-15277d2e2g5+10894be3g5-15659ce3g5\
   +7347de3g5-15091e4g5+11972c3fg5+13378bcdfg5-9263c2dfg5+15607bd2fg5-8082cd\
   2fg5-7204d3fg5-5155bcefg5+13145c2efg5+11459bdefg5+15001cdefg5+12534d2efg5\
   +871be2fg5-6596ce2fg5-15178de2fg5-5983e3fg5-6980bcf2g5-5714c2f2g5+7419bdf\
   2g5-6289cdf2g5+8031d2f2g5-4943bef2g5-15683cef2g5-3184def2g5+122e2f2g5-158\
   07bf3g5+612cf3g5+15002df3g5-2845ef3g5-2796f4g5-1888c3g6+13438bcdg6+6443c2\
   dg6+10263bd2g6+14035cd2g6-3644d3g6+13626bceg6-7266c2eg6+14423bdeg6-6722cd\
   eg6+11229d2eg6+7353be2g6+14928ce2g6-9430de2g6-5801e3g6+4674bcfg6-10619c2f\
   g6+12741bdfg6-14521cdfg6-1532d2fg6+9475befg6+5056cefg6-2133defg6-9752e2fg\
   6-13732bf2g6+8140cf2g6-3526df2g6+14854ef2g6+12576f3g6-13560bcg7-5702c2g7+\
   5826bdg7-13691cdg7-4085d2g7+7307beg7+6529ceg7-5622deg7-3069e2g7+11568bfg7\
   -7185cfg7-11642dfg7+8477efg7-4397f2g7+6873bg8-9094cg8+1881dg8-6773eg8+115\
   91fg8-11281g9-12602bc+12694c2+2462bd+14545cd-11898d2+11888be+13372ce+6755\
   de+6418e2+14931bf-1563cf-9863df+6098ef+2463f2-10606bg+6483cg-12434dg-1560\
   7eg+7841fg+7846g2
↦ j[95]=bd2f2g4-139e3f2g4-8666bcf3g4-4667c2f3g4-2285bdf3g4+971cdf3g4+5259d2\
   f3g4-9919bef3g4-958cef3g4+3168def3g4+125e2f3g4+14489bf4g4-12100cf4g4+8731\
   df4g4+9315ef4g4-5921f5g4+4313c4g5-8424bd3g5-5325cd3g5+7243d4g5-15587c3eg5\
   +7693bcdeg5+9838c2deg5+1259bd2eg5+10915cd2eg5-4131d3eg5+5249bce2g5-11861c\
   2e2g5-2601bde2g5-6308cde2g5-12016d2e2g5+11245be3g5+15564ce3g5+1430de3g5-9\
   180e4g5+12346c3fg5+12219bcdfg5+15195c2dfg5+12039bd2fg5-6649cd2fg5-10275d3\
   fg5+15906bcefg5+11120c2efg5-10566bdefg5+13155cdefg5-9026d2efg5+2392be2fg5\
   -5716ce2fg5-3216de2fg5+2452e3fg5-7878bcf2g5-12417c2f2g5-12299bdf2g5-9884c\
   df2g5+15802d2f2g5+2320bef2g5+3053cef2g5-14981def2g5-526e2f2g5+15872bf3g5-\
   10753cf3g5-2746df3g5-3013ef3g5+9806f4g5+9806c3g6+14523bcdg6-13990c2dg6+43\
   78bd2g6-2457cd2g6+6691d3g6-13675bceg6+8480c2eg6+10091bdeg6-2440cdeg6+6413\
   d2eg6+15602be2g6-7833ce2g6+12112de2g6+5685e3g6-947bcfg6-7956c2fg6-14208bd\
   fg6+8495cdfg6-8494d2fg6-7295befg6+7156cefg6+12876defg6-14906e2fg6-9352bf2\
   g6+2163cf2g6+10797df2g6+4141ef2g6-10570f3g6+2363bcg7-8580c2g7-9592bdg7+14\
   365cdg7+8977d2g7-8712beg7+5538ceg7-3657deg7+11787e2g7-3414bfg7-1830cfg7+7\
   858dfg7-15058efg7+12212f2g7+15616bg8+13837cg8+1142dg8+14328eg8-7581fg8-31\
   56g9+11664bc-6711c2-12536bd-4939cd+10012d2+7452be+4664ce+9485de-14397e2+7\
   878bf+10815cf+11649df-7850ef+6521f2-15879bg-1275cg+10010dg-15001eg-43fg-7\
   691g2
↦ j[96]=c2df2g4+5171e3f2g4+3477bcf3g4-2794c2f3g4-2615bdf3g4-13656cdf3g4+136\
   26d2f3g4+5589bef3g4-10349cef3g4-13438def3g4+1285e2f3g4-9532bf4g4-14694cf4\
   g4-4622df4g4-4990ef4g4+8191f5g4-15619c4g5-13671bd3g5-13958cd3g5+14104d4g5\
   +4755c3eg5+9015bcdeg5+13166c2deg5+13939bd2eg5+9246cd2eg5-15556d3eg5+14177\
   bce2g5-5935c2e2g5-63bde2g5+5266cde2g5-4808d2e2g5+8725be3g5+4576ce3g5+9800\
   de3g5-11183e4g5-12506c3fg5-6193bcdfg5+962c2dfg5-2344bd2fg5+7038cd2fg5+111\
   31d3fg5-7276bcefg5+6541c2efg5-11794bdefg5-3722cdefg5+2579d2efg5+7484be2fg\
   5-8780ce2fg5-3199de2fg5+5030e3fg5-12396bcf2g5-6051c2f2g5+16000bdf2g5+7050\
   cdf2g5+2550d2f2g5+4759bef2g5+12534cef2g5-9249def2g5+10691e2f2g5+6705bf3g5\
   +4244cf3g5+1167df3g5-4614ef3g5+4471f4g5+11331c3g6-6171bcdg6-1366c2dg6+236\
   2bd2g6+12794cd2g6+11811d3g6+14013bceg6+1284c2eg6+6416bdeg6+7447cdeg6-6581\
   d2eg6+9119be2g6-10297ce2g6-13257de2g6+15336e3g6+15166bcfg6-14064c2fg6+130\
   42bdfg6-4564cdfg6+15717d2fg6+2246befg6-9413cefg6+12978defg6-10038e2fg6+14\
   631bf2g6-8659cf2g6-7718df2g6-12453ef2g6-5993f3g6-1349bcg7-14920c2g7-10131\
   bdg7-5375cdg7-1484d2g7+11201beg7+10597ceg7-12191deg7+13472e2g7-11256bfg7-\
   14118cfg7-11023dfg7+573efg7-7289f2g7+1889bg8-2220cg8+10742dg8-7167eg8-581\
   8fg8+5925g9-7863bc+12443c2+11570bd-8325cd+4698d2+1631be+1866ce-8543de-871\
   3e2-7567bf-15266cf-11506df-2855ef-11886f2+13074bg-1455cg+9336dg+14665eg+9\
   603fg-9522g2
↦ j[97]=bcdf2g4-6502e3f2g4-8905bcf3g4-12219c2f3g4-4881bdf3g4+15479cdf3g4+31\
   2d2f3g4+9405bef3g4+11163cef3g4+642def3g4-9057e2f3g4+1785bf4g4+1906cf4g4+5\
   261df4g4-7347ef4g4-5393f5g4-14761c4g5+15197bd3g5+14504cd3g5-8976d4g5+2923\
   c3eg5-3450bcdeg5-8292c2deg5-10751bd2eg5+15627cd2eg5-2874d3eg5-12289bce2g5\
   +1853c2e2g5-6244bde2g5-3868cde2g5+6374d2e2g5+13721be3g5+11808ce3g5+14729d\
   e3g5+5038e4g5+8375c3fg5-3688bcdfg5-6699c2dfg5-2593bd2fg5+8700cd2fg5+7660d\
   3fg5+2746bcefg5+14684c2efg5-15891bdefg5-808cdefg5-9525d2efg5+13515be2fg5-\
   13668ce2fg5+14719de2fg5-3695e3fg5-2789bcf2g5-14380c2f2g5-2249bdf2g5-10888\
   cdf2g5+6445d2f2g5+5483bef2g5-15497cef2g5-4188def2g5+7803e2f2g5+73bf3g5-86\
   02cf3g5+8406df3g5+5665ef3g5-6781f4g5+3671c3g6+3986bcdg6+14702c2dg6-11769b\
   d2g6+15890cd2g6+13147d3g6-11181bceg6-9804c2eg6-7435bdeg6-2866cdeg6+14285d\
   2eg6-1290be2g6-6669ce2g6-9874de2g6+3162e3g6+304bcfg6-8550c2fg6+6594bdfg6-\
   9807cdfg6-1989d2fg6-12607befg6+2705cefg6+12162defg6+243e2fg6+4131bf2g6+95\
   48cf2g6-2808df2g6-13321ef2g6-420f3g6-853bcg7+1386c2g7-7602bdg7-4091cdg7+1\
   3866d2g7-7821beg7+5947ceg7-10998deg7+14183e2g7-2951bfg7-2541cfg7+13289dfg\
   7+12306efg7+5625f2g7+14995bg8-4849cg8+11214dg8-6995eg8-6009fg8+5482g9-116\
   07bc+14660c2-11549bd+15883cd+8619d2-12150be+4566ce+5489de-10967e2+12366bf\
   -3355cf-3532df-10742ef-14151f2+4237bg-6360cg+5889dg+4313eg-14915fg+13334g\
   2
↦ j[98]=c3f2g4-3643e3f2g4-4316bcf3g4+8257c2f3g4+803bdf3g4+10806cdf3g4-12705\
   d2f3g4-2694bef3g4-10297cef3g4+12338def3g4+9066e2f3g4+9435bf4g4-2834cf4g4-\
   13403df4g4+14208ef4g4+2548f5g4-12889c4g5+88bd3g5-2545cd3g5-12719d4g5-6991\
   c3eg5-6046bcdeg5-3857c2deg5-15606bd2eg5-3588cd2eg5-6362d3eg5-8889bce2g5-1\
   0747c2e2g5+10300bde2g5+1967cde2g5+8375d2e2g5-12007be3g5+8227ce3g5+14631de\
   3g5-8916e4g5-6899c3fg5-3929bcdfg5-11273c2dfg5-13819bd2fg5+3645cd2fg5+1422\
   5d3fg5+11465bcefg5-13733c2efg5+10339bdefg5-1613cdefg5-14816d2efg5+9502be2\
   fg5-7248ce2fg5+3567de2fg5-8640e3fg5-7894bcf2g5+9841c2f2g5-542bdf2g5+11056\
   cdf2g5-12319d2f2g5+15516bef2g5-7191cef2g5+6661def2g5+3320e2f2g5+2612bf3g5\
   -12175cf3g5-3742df3g5-12256ef3g5+2213f4g5-10149c3g6+4025bcdg6+1875c2dg6-1\
   746bd2g6+6959cd2g6+2658d3g6+13560bceg6-9121c2eg6+2936bdeg6-6639cdeg6-8424\
   d2eg6-7683be2g6+4953ce2g6-6998de2g6+8301e3g6-9469bcfg6-15987c2fg6+325bdfg\
   6+2351cdfg6+6458d2fg6+6000befg6-902cefg6+12897defg6+9315e2fg6+9397bf2g6+1\
   4115cf2g6+11978df2g6-2135ef2g6-11872f3g6-4706bcg7+12495c2g7+2444bdg7-5065\
   cdg7-7551d2g7+7484beg7-4777ceg7-13500deg7-11557e2g7+10204bfg7-5762cfg7+90\
   57dfg7+11984efg7+14868f2g7+15897bg8+437cg8+15002dg8+8250eg8+1579fg8+7303g\
   9-13053bc+4611c2-10217bd+10614cd+8994d2-9249be-6448ce-13158de+12283e2-779\
   6bf-15576cf-4359df-5023ef-794f2-2671bg-4643cg+6483dg-1925eg+594fg+12397g2
↦ j[99]=cde2g6-5663d2e2g6-57be3g6-7502ce3g6-6956de3g6-2489e4g6-13152c3fg6-7\
   451bcdfg6-6327c2dfg6-7484bd2fg6-7464cd2fg6+15676d3fg6+21bcefg6+6818c2efg6\
   +14284bdefg6+4245cdefg6-4781d2efg6+8151be2fg6+1385ce2fg6-10983de2fg6+11e3\
   fg6+8784bcf2g6+5155c2f2g6+12775bdf2g6+11990cdf2g6-6310d2f2g6-5957bef2g6+6\
   71cef2g6+10396def2g6+12733e2f2g6-11026bf3g6+6915cf3g6+9045df3g6-8397ef3g6\
   +9923f4g6-4885c3g7+3164bcdg7-13623c2dg7+2315bd2g7-8713cd2g7+2492d3g7-2648\
   bceg7+9682c2eg7+6360bdeg7+10056cdeg7-10947d2eg7-12593be2g7+431ce2g7-7569d\
   e2g7+9888e3g7+12336bcfg7-15061c2fg7-306bdfg7-6650cdfg7+7022d2fg7+3612befg\
   7+11419cefg7+15681defg7+13319e2fg7-7456bf2g7-13941cf2g7+4854df2g7-12116ef\
   2g7-15315f3g7+15601bcg8-759c2g8+10642bdg8+1421cdg8-3035d2g8-3975beg8+706c\
   eg8-2304deg8-14648e2g8-15571bfg8-1976cfg8+7457dfg8-14433efg8+10138f2g8-32\
   39bg9-903cg9+4074dg9+9924eg9+7286fg9-5777g10-3619c3-12940bcd-2717c2d+1110\
   7bd2+7421cd2-1454d3-10186bce+10747c2e-8222bde+12181cde-291d2e+3327be2-147\
   83ce2+8784de2-5500e3-15518bcf-8114c2f+6712bdf+8466cdf+8794d2f-1701bef+358\
   1cef-13195def+11129e2f+5558bf2-1219cf2+5217df2-13640ef2-6072f3+13811bcg-3\
   445c2g+12747bdg+4271cdg+12470d2g-10062beg-15443ceg+12214deg+1124e2g-4019b\
   fg+12149cfg-7610dfg-6749efg+11758f2g-10896bg2-8204cg2-8435dg2+9853eg2-647\
   4fg2-1330g3
↦ j[100]=bde2g6-14848d2e2g6+6394be3g6+1887ce3g6+15822de3g6-3345e4g6-6466c3f\
   g6+9047bcdfg6-6145c2dfg6+3206bd2fg6+8304cd2fg6+8988d3fg6+12835bcefg6+976c\
   2efg6-7183bdefg6+1161cdefg6+3027d2efg6-5443be2fg6+361ce2fg6-1707de2fg6-85\
   87e3fg6+844bcf2g6+13579c2f2g6+14688bdf2g6+271cdf2g6+9259d2f2g6+10100bef2g\
   6+11321cef2g6-14106def2g6-12764e2f2g6-7984bf3g6+6123cf3g6-10349df3g6+1469\
   3ef3g6+13859f4g6-5016c3g7+14230bcdg7+8220c2dg7-14597bd2g7+9596cd2g7+13190\
   d3g7-4787bceg7-9530c2eg7-4061bdeg7+11802cdeg7+9702d2eg7+309be2g7-11697ce2\
   g7-8113de2g7+4378e3g7-12479bcfg7-9185c2fg7+7489bdfg7+10700cdfg7-3953d2fg7\
   -8999befg7+3669cefg7-7677defg7+523e2fg7-2679bf2g7+6840cf2g7-6092df2g7-462\
   8ef2g7+11955f3g7-452bcg8+7978c2g8+15037bdg8+11730cdg8+5994d2g8-1493beg8+2\
   301ceg8-14000deg8+8477e2g8+8441bfg8-1703cfg8-4682dfg8-225efg8+4373f2g8+14\
   908bg9+7533cg9-4756dg9-4698eg9+12308fg9+165g10+15450c3-5976bcd+8364c2d-47\
   82bd2+3498cd2+14734d3+11197bce-8121c2e-14445bde-8375cde-5200d2e+12363be2-\
   6673ce2+12972de2-13849e3+8461bcf-6440c2f-6589bdf+1781cdf-14854d2f+4980bef\
   -1300cef+13223def+6862e2f+6146bf2+8012cf2-11906df2-4811ef2+10224f3-2546bc\
   g+14547c2g-1324bdg+11885cdg+11764d2g-3860beg-2123ceg-13183deg-2246e2g+152\
   31bfg+15538cfg+2369dfg+13928efg+7474f2g-2222bg2+3182cg2+7946dg2+976eg2+13\
   322fg2+6255g3
↦ j[101]=c2e2g6-6671d2e2g6-5649be3g6-2556ce3g6-5502de3g6-6083e4g6-14827c3fg\
   6+12901bcdfg6-4837c2dfg6-9497bd2fg6+2419cd2fg6-1839d3fg6+15267bcefg6-8275\
   c2efg6-4163bdefg6-7541cdefg6-15142d2efg6-8297be2fg6-4094ce2fg6-3381de2fg6\
   +5694e3fg6+3910bcf2g6-3480c2f2g6+10422bdf2g6+11699cdf2g6+8531d2f2g6-3188b\
   ef2g6-15483cef2g6-11099def2g6-13570e2f2g6-5899bf3g6-5577cf3g6+4731df3g6-9\
   134ef3g6-11617f4g6-3167c3g7-8976bcdg7-14915c2dg7+5970bd2g7-4081cd2g7+1454\
   9d3g7-8278bceg7-11103c2eg7+14776bdeg7-7187cdeg7+6885d2eg7+15201be2g7+9375\
   ce2g7+11733de2g7-5547e3g7-12066bcfg7-12428c2fg7-4059bdfg7+14783cdfg7+4370\
   d2fg7+6620befg7+14499cefg7+8575defg7-15224e2fg7-15053bf2g7-3470cf2g7-1780\
   df2g7-10629ef2g7+7051f3g7-7083bcg8+15035c2g8-12374bdg8+13371cdg8-7486d2g8\
   +3775beg8+7900ceg8-2185deg8+10584e2g8-3640bfg8+1817cfg8+12730dfg8+5369efg\
   8-2281f2g8-2385bg9-12068cg9+5199dg9-429eg9+7236fg9-11919g10+5633c3-3803bc\
   d-4102c2d+10288bd2+210cd2+6187d3-12759bce-9171c2e+4420bde+13786cde-7964d2\
   e+5766be2-14543ce2-2947de2-832e3-8737bcf-1431c2f+15938bdf+6123cdf+9485d2f\
   +7431bef-7026cef+8749def+13812e2f+4496bf2+5990cf2+6080df2-1139ef2-7816f3+\
   1031bcg+8672c2g-10727bdg+11461cdg-8180d2g+11402beg-8098ceg+14585deg-3781e\
   2g-15199bfg-8425cfg+1674dfg-5349efg+3858f2g+12218bg2-14206cg2+9257dg2+681\
   2eg2-14237fg2-4662g3
↦ j[102]=bce2g6+13346d2e2g6-11083be3g6+14764ce3g6+1852de3g6+10305e4g6-8038c\
   3fg6-1091bcdfg6-8366c2dfg6-5695bd2fg6-5363cd2fg6+13064d3fg6+10694bcefg6+1\
   2047c2efg6+11008bdefg6+4300cdefg6+7041d2efg6-10819be2fg6-3537ce2fg6+13838\
   de2fg6+10402e3fg6+7353bcf2g6+11410c2f2g6-8657bdf2g6+1006cdf2g6-4231d2f2g6\
   -342bef2g6-14709cef2g6-14844def2g6+5612e2f2g6+13797bf3g6+13526cf3g6+5588d\
   f3g6-15821ef3g6+796f4g6+3197c3g7+8813bcdg7+12182c2dg7+7725bd2g7+8759cd2g7\
   -4509d3g7+11142bceg7-8983c2eg7+12522bdeg7+4018cdeg7-14998d2eg7+4294be2g7-\
   4812ce2g7-6394de2g7+9747e3g7-8091bcfg7-998c2fg7+3466bdfg7-15403cdfg7+1081\
   8d2fg7+13409befg7+10372cefg7+15662defg7+4439e2fg7-6263bf2g7+5095cf2g7+147\
   4df2g7-13126ef2g7+7052f3g7-10159bcg8-13647c2g8-1145bdg8-7010cdg8+7948d2g8\
   -12942beg8+2035ceg8+2066deg8-7894e2g8+9504bfg8+10180cfg8+543dfg8-1329efg8\
   -8365f2g8+11142bg9-15365cg9+3824dg9+833eg9-10721fg9-14953g10-8259c3+8682b\
   cd+13176c2d+11064bd2-10355cd2-12721d3+1872bce+7681c2e-14784bde-6739cde+12\
   532d2e+13436be2-11903ce2+6053de2-7035e3+8882bcf-13971c2f-5481bdf+5426cdf-\
   10604d2f+2892bef-2609cef-5666def-8196e2f+15457bf2+8058cf2+7760df2+5763ef2\
   -6193f3+7029bcg+14199c2g-739bdg-15391cdg-10027d2g+3418beg+13921ceg+4068de\
   g+11063e2g+12692bfg-9184cfg+13299dfg+7115efg+14212f2g+15009bg2-2196cg2-11\
   505dg2+9635eg2+5910fg2-11039g3
↦ j[103]=d3eg6-11902d2e2g6+1819be3g6+15898ce3g6+1616de3g6+5309e4g6-14601c3f\
   g6+14528bcdfg6+6773c2dfg6-6741bd2fg6-95cd2fg6+11793d3fg6+15103bcefg6-2657\
   c2efg6+9969bdefg6-1984cdefg6-7551d2efg6-15228be2fg6+6668ce2fg6-10873de2fg\
   6-15487e3fg6-10739bcf2g6+2543c2f2g6+2598bdf2g6-629cdf2g6+3756d2f2g6+1268b\
   ef2g6+1486cef2g6+2550def2g6+1143e2f2g6-8730bf3g6-6038cf3g6-1440df3g6+9857\
   ef3g6-2357f4g6-10412c3g7-3736bcdg7-5342c2dg7+5508bd2g7-10083cd2g7-5796d3g\
   7+12274bceg7-8879c2eg7+1470bdeg7-9445cdeg7-2492d2eg7-7961be2g7+11066ce2g7\
   -4260de2g7-4773e3g7-3708bcfg7-15711c2fg7+420bdfg7-2753cdfg7+10108d2fg7+52\
   12befg7+2611cefg7+4430defg7-429e2fg7+14046bf2g7+7712cf2g7-15000df2g7-1586\
   7ef2g7+10523f3g7+15010bcg8+11030c2g8-14963bdg8+9045cdg8+10452d2g8-5478beg\
   8-1089ceg8-13584deg8+15875e2g8+6039bfg8+15435cfg8-9489dfg8-8410efg8+13210\
   f2g8+13980bg9+13993cg9+3829dg9-4818eg9+13990fg9+2180g10+782c3-2877bcd+651\
   1c2d+14526bd2+1794cd2-2434d3-9141bce-4411c2e+9740bde+13605cde-12085d2e-30\
   87be2+8839ce2+11595de2+7366e3-2108bcf+4892c2f+11089bdf+1528cdf+1154d2f+15\
   613bef-6060cef-15645def+5167e2f+15658bf2+13228cf2+15090df2+664ef2+5341f3+\
   7395bcg+4334c2g+8148bdg-1685cdg+9031d2g+11219beg-4471ceg-14996deg-9497e2g\
   +93bfg+7807cfg+5514dfg-11112efg+12256f2g+9784bg2+10533cg2+12296dg2+2006eg\
   2-6412fg2-9155g3
↦ j[104]=cd2eg6-4544d2e2g6-4919be3g6+15867ce3g6-9487de3g6-9882e4g6-58c3fg6-\
   3051bcdfg6-12249c2dfg6-5734bd2fg6-2099cd2fg6+13993d3fg6+4776bcefg6-10431c\
   2efg6+2375bdefg6-2465cdefg6+11152d2efg6+6873be2fg6-14020ce2fg6+7849de2fg6\
  +11745e3fg6-6026bcf2g6+239c2f2g6-12481bdf2g6+9271cdf2g6+586d2f2g6-4751bef\
  2g6+6430cef2g6-1122def2g6+1508e2f2g6-11324bf3g6-5940cf3g6+10556df3g6+2568\
  ef3g6-9519f4g6-10905c3g7+12924bcdg7-2554c2dg7-6730bd2g7+2102cd2g7-11037d3\
  g7-13958bceg7-14048c2eg7-229bdeg7+14527cdeg7-9865d2eg7+1189be2g7-7181ce2g\
  7+5277de2g7+2120e3g7+5944bcfg7+2601c2fg7+11454bdfg7+7936cdfg7-12987d2fg7-\
  11811befg7+6746cefg7+1401defg7+14424e2fg7-1427bf2g7+11806cf2g7-12068df2g7\
  +2599ef2g7-10140f3g7-14061bcg8+11113c2g8-7986bdg8-5050cdg8-12863d2g8-1574\
  3beg8+7723ceg8-1942deg8-2193e2g8-13322bfg8+3733cfg8+11164dfg8-10069efg8-8\
  84f2g8+672bg9-4879cg9-8097dg9+1284eg9+14076fg9-11759g10-8099c3+9750bcd+13\
  478c2d+8622bd2+15727cd2-10027d3+11405bce-6633c2e+12683bde-2224cde-13435d2\
  e+2096be2+9594ce2+12378de2-10652e3+14711bcf+6653c2f+7030bdf-6990cdf+14142\
  d2f+4627bef+15851cef-2988def-4476e2f+11971bf2+13844cf2-14454df2+14549ef2-\
  3454f3-3488bcg-13488c2g-12998bdg+2697cdg+12124d2g-11857beg-14830ceg-5729d\
  eg+12039e2g-3318bfg-9304cfg+7dfg+13310efg+11569f2g+4109bg2-12930cg2+2714d\
  g2-8847eg2+10746fg2-14922g3
↦ j[105]=bd2eg6+2692d2e2g6+594be3g6+14555ce3g6+4784de3g6-4284e4g6+7384c3fg6\
  +6371bcdfg6+3970c2dfg6-13154bd2fg6+160cd2fg6-5228d3fg6+9861bcefg6-6910c2e\
  fg6-8086bdefg6+5038cdefg6+705d2efg6+2992be2fg6+4782ce2fg6+10346de2fg6-510\
  5e3fg6-15043bcf2g6+12420c2f2g6+6733bdf2g6+2706cdf2g6+7344d2f2g6-15239bef2\
  g6-10653cef2g6+120def2g6-4964e2f2g6-14056bf3g6-9894cf3g6-6974df3g6-14189e\
  f3g6-12093f4g6-13602c3g7+15824bcdg7-2499c2dg7-7298bd2g7+2864cd2g7-3246d3g\
  7+14391bceg7-8046c2eg7+1803bdeg7-11394cdeg7-7810d2eg7-12163be2g7-9281ce2g\
  7+12141de2g7+15900e3g7-3369bcfg7+10988c2fg7+12856bdfg7-8889cdfg7+9840d2fg\
  7-14914befg7+6180cefg7+13541defg7+15701e2fg7+6140bf2g7-9564cf2g7-4508df2g\
  7+13076ef2g7-2474f3g7-586bcg8-15921c2g8+3915bdg8-10973cdg8+1025d2g8+9242b\
  eg8-12698ceg8-5711deg8-5919e2g8+12323bfg8+4362cfg8+3745dfg8-8303efg8-150f\
  2g8+10769bg9+15607cg9-15331dg9+11409eg9+7631fg9+4242g10+7464c3-15387bcd+1\
  4776c2d-10376bd2+8035cd2-12094d3-14823bce-3842c2e+9504bde-971cde+6315d2e-\
  5949be2-14ce2-1474de2+1889e3+3410bcf+10415c2f-1920bdf-1848cdf+370d2f+1432\
  4bef-12999cef-8684def+15093e2f+13978bf2-13866cf2+4614df2+5881ef2+758f3+65\
  85bcg-3133c2g-11971bdg+7059cdg-1785d2g-4247beg-14595ceg-7801deg-15242e2g+\
  14696bfg-5575cfg+8247dfg-13079efg+8883f2g-7731bg2-10191cg2-11132dg2-10401\
  eg2-11107fg2+7383g3
↦ j[106]=c2deg6-13878d2e2g6-9069be3g6-1407ce3g6-11760de3g6-5409e4g6+10943c3\
  fg6+9660bcdfg6-2325c2dfg6-8082bd2fg6-15242cd2fg6-2576d3fg6+7336bcefg6+148\
  02c2efg6+5639bdefg6+3174cdefg6-2702d2efg6+2554be2fg6+7985ce2fg6-13321de2f\
  g6+15841e3fg6+12857bcf2g6+13616c2f2g6+13533bdf2g6-6777cdf2g6-6455d2f2g6-1\
  2385bef2g6+8539cef2g6-5767def2g6-7965e2f2g6+9619bf3g6+3862cf3g6+15458df3g\
  6-8002ef3g6+15855f4g6+8838c3g7+6050bcdg7-956c2dg7+11746bd2g7+4568cd2g7-11\
  656d3g7+13313bceg7+4157c2eg7+12381bdeg7-9855cdeg7+14993d2eg7-8945be2g7-31\
  6ce2g7-8689de2g7+4829e3g7-4022bcfg7-10335c2fg7+4528bdfg7-13985cdfg7+1415d\
  2fg7+10659befg7+13748cefg7+9691defg7+3073e2fg7-11456bf2g7-2111cf2g7+314df\
  2g7+1515ef2g7+9097f3g7+2267bcg8-5287c2g8+2426bdg8+15027cdg8-10216d2g8-166\
  3beg8+6089ceg8+6098deg8+11687e2g8-13992bfg8+963cfg8+1336dfg8+5756efg8-535\
  9f2g8+8465bg9-3180cg9-4523dg9+8705eg9+12845fg9+5130g10+9606c3-1895bcd+824\
  4c2d+6192bd2-9671cd2+2309d3-2905bce+8933c2e-3070bde-13265cde-6169d2e+1306\
  6be2-487ce2-4486de2-11786e3+11322bcf+10319c2f-5746bdf-14076cdf-4613d2f-20\
  13bef-7313cef-9112def+12449e2f+10120bf2-5987cf2+8206df2+12235ef2+7983f3+1\
  5195bcg-14458c2g-12251bdg-5817cdg+15993d2g+7201beg-13594ceg-2137deg-13827\
  e2g+2099bfg-12229cfg-12736dfg-3378efg-11118f2g-4725bg2+6500cg2+9457dg2+13\
  526eg2+3143fg2+9463g3
↦ j[107]=bcdeg6-529d2e2g6+1422be3g6-7458ce3g6-4888de3g6-10735e4g6+7279c3fg6\
  +8024bcdfg6+6576c2dfg6+3232bd2fg6-10634cd2fg6-6122d3fg6+1843bcefg6-14875c\
   2efg6+14496bdefg6-1843cdefg6-11351d2efg6+1918be2fg6+2932ce2fg6-10384de2fg\
   6-2130e3fg6+7225bcf2g6-1535c2f2g6-3026bdf2g6+14115cdf2g6-4609d2f2g6-11009\
   bef2g6-10778cef2g6+6124def2g6+13650e2f2g6+12832bf3g6-13815cf3g6-2236df3g6\
   +6259ef3g6+10744f4g6-8081c3g7+1943bcdg7-322c2dg7+8632bd2g7+2503cd2g7-6079\
   d3g7+393bceg7-11639c2eg7+7658bdeg7+359cdeg7-1508d2eg7+14155be2g7-6477ce2g\
   7+9986de2g7+10211e3g7-4339bcfg7+10193c2fg7-13327bdfg7-3580cdfg7-13850d2fg\
   7-1710befg7-703cefg7+1687defg7-2969e2fg7+12597bf2g7-9920cf2g7+5468df2g7-5\
   440ef2g7+2401f3g7-99bcg8+15328c2g8+8808bdg8+15503cdg8+8672d2g8-15250beg8+\
   4169ceg8-4365deg8-999e2g8+6894bfg8+11827cfg8-8167dfg8+10937efg8+3239f2g8+\
   5258bg9-11611cg9+9200dg9-10230eg9-15553fg9-5405g10-7833c3-15305bcd+10920c\
   2d+15566bd2-7052cd2+8440d3+3461bce+197c2e+13489bde+9787cde-14772d2e-2912b\
   e2+1334ce2+10377de2+7637e3+5530bcf-15583c2f+9592bdf+15430cdf-467d2f-2301b\
   ef+2657cef-7802def-14883e2f-12446bf2-15594cf2+15057df2+11465ef2+4901f3-26\
   80bcg+4200c2g+2858bdg-11890cdg-14560d2g-15311beg-5856ceg-3279deg+9404e2g-\
   12255bfg-8897cfg+14315dfg-6960efg-579f2g+8149bg2+8011cg2-8231dg2+5925eg2-\
   10386fg2+2019g3
↦ j[108]=c3eg6-10838d2e2g6-12898be3g6+9189ce3g6-6921de3g6+8069e4g6-8407c3fg\
   6-2120bcdfg6+5728c2dfg6-7254bd2fg6-7584cd2fg6-11015d3fg6+11266bcefg6-4450\
   c2efg6-3252bdefg6+5814cdefg6+14097d2efg6-11763be2fg6-1125ce2fg6+7719de2fg\
   6-8823e3fg6+13241bcf2g6-9585c2f2g6-14058bdf2g6+2148cdf2g6-319d2f2g6-9963b\
   ef2g6-6617cef2g6-2394def2g6+8898e2f2g6-3442bf3g6-14824cf3g6+10956df3g6+97\
   58ef3g6-13390f4g6-6784c3g7+5622bcdg7+15881c2dg7+959bd2g7+3363cd2g7+837d3g\
   7-869bceg7+15329c2eg7-14141bdeg7+6161cdeg7-9860d2eg7+221be2g7-15456ce2g7-\
   11705de2g7-7095e3g7+962bcfg7+8130c2fg7+15949bdfg7+10470cdfg7-12896d2fg7+1\
   4180befg7+45cefg7+5749defg7-10683e2fg7+7756bf2g7-3390cf2g7+3913df2g7-1783\
   ef2g7+12302f3g7-7429bcg8-12546c2g8+9975bdg8-9333cdg8+2674d2g8-10616beg8-6\
   845ceg8+1874deg8-12925e2g8+11122bfg8-5192cfg8-4806dfg8-11725efg8+5652f2g8\
   -10292bg9-13725cg9-4480dg9+1492eg9+15504fg9-14336g10-6921c3+6867bcd+2050c\
   2d+7130bd2+10013cd2+1315d3-10327bce+11370c2e-1721bde-2842cde+15946d2e+146\
   15be2+426ce2+4245de2-6108e3-12901bcf-15249c2f+13141bdf-10225cdf+10235d2f+\
   833bef-673cef+5938def+4387e2f-9634bf2+9643cf2-578df2+8707ef2-11936f3+2936\
   bcg-1845c2g+11665bdg+12788cdg-11715d2g+7869beg+6307ceg+4247deg-9216e2g-15\
   607bfg-7936cfg-3570dfg-10940efg-3481f2g+1978bg2-13179cg2-7245dg2-7681eg2-\
   2474fg2-11711g3
↦ j[109]=d4g6+2905d2e2g6+9942be3g6+11488ce3g6+14675de3g6+7385e4g6-11150c3fg\
   6-13553bcdfg6+15645c2dfg6+3613bd2fg6+534cd2fg6+12743d3fg6-4364bcefg6-1091\
   0c2efg6+9312bdefg6-7220cdefg6-5311d2efg6+7235be2fg6+7302ce2fg6+652de2fg6+\
   12195e3fg6+6781bcf2g6-11433c2f2g6+3821bdf2g6+830cdf2g6+7016d2f2g6-7458bef\
   2g6+3394cef2g6+6541def2g6+10405e2f2g6+14029bf3g6-3672cf3g6+10135df3g6+108\
   15ef3g6+1605f4g6-4068c3g7+261bcdg7-680c2dg7+15722bd2g7+3861cd2g7-8323d3g7\
   -14015bceg7+15865c2eg7-11129bdeg7+1373cdeg7-11162d2eg7+14803be2g7-12797ce\
   2g7+1047de2g7+4900e3g7+13344bcfg7-11098c2fg7-15946bdfg7+12383cdfg7-14425d\
   2fg7+1307befg7-942cefg7-1400defg7+15284e2fg7+7928bf2g7-11565cf2g7-15197df\
   2g7+9456ef2g7+11380f3g7+2136bcg8+15970c2g8-4838bdg8-13936cdg8-4237d2g8+89\
   36beg8-56ceg8-8869deg8-5662e2g8-14704bfg8-9383cfg8-6583dfg8+7031efg8-3435\
   f2g8+1744bg9+9872cg9+9036dg9+4983eg9+6709fg9+7852g10-1992c3+13677bcd-33c2\
   d+6755bd2+14907cd2+3202d3+12962bce-13787c2e-784bde+12711cde+2801d2e-7222b\
   e2+10275ce2-8352de2+15288e3-8574bcf-7603c2f+15097bdf-9709cdf+1658d2f-1529\
   9bef-6060cef-2546def-2095e2f+4228bf2-9350cf2+1742df2-4024ef2-2075f3+9731b\
   cg-10387c2g+10128bdg-5880cdg+6332d2g+12940beg+13848ceg+10138deg-9685e2g+7\
   653bfg+5241cfg+3731dfg-5779efg+15606f2g-190bg2-3077cg2+11171dg2-4050eg2+4\
   146fg2-10058g3
↦ j[110]=cd3g6+8069d2e2g6+2172be3g6+9608ce3g6-10519de3g6+2434e4g6+5663c3fg6\
   +14642bcdfg6-15484c2dfg6-11593bd2fg6-7785cd2fg6+11274d3fg6+8773bcefg6-146\
   99c2efg6-9100bdefg6+14555cdefg6-1101d2efg6+14884be2fg6+4186ce2fg6-14928de\
   2fg6-14540e3fg6+14608bcf2g6+13891c2f2g6+4802bdf2g6+15898cdf2g6+14307d2f2g\
   6+6510bef2g6+12015cef2g6-11787def2g6-4245e2f2g6+1339bf3g6+7509cf3g6-2837d\
   f3g6-4624ef3g6-13264f4g6-4157c3g7-12090bcdg7-11640c2dg7+4020bd2g7-13397cd\
   2g7-13588d3g7+5208bceg7-1926c2eg7+15399bdeg7+5462cdeg7-1576d2eg7+3125be2g\
   7+15369ce2g7-11738de2g7+12859e3g7-11454bcfg7+8238c2fg7+9161bdfg7-7011cdfg\
   7-8131d2fg7-10469befg7+4834cefg7+373defg7+14164e2fg7-2895bf2g7+12266cf2g7\
   +6511df2g7-3720ef2g7-9140f3g7+8700bcg8+12518c2g8-4475bdg8-11629cdg8-8447d\
   2g8-11505beg8-5899ceg8+7257deg8-9042e2g8+6859bfg8-983cfg8+11928dfg8+12935\
   efg8-7074f2g8+1436bg9+8254cg9-9069dg9+5548eg9+13024fg9-14059g10+4938c3-14\
   530bcd-4209c2d+11928bd2-5071cd2-3970d3-7581bce+1016c2e+9856bde+15964cde+5\
   52d2e-9442be2+10873ce2+12498de2-13812e3-14397bcf+13508c2f+11348bdf+34cdf+\
   12464d2f+12513bef+7004cef-13025def+6992e2f+5678bf2-13576cf2+12047df2+7184\
   ef2-1547f3-9162bcg+7280c2g+13832bdg-14988cdg+12365d2g+2998beg+8240ceg-373\
   deg-11829e2g-11427bfg-12619cfg-10429dfg-14844efg-9172f2g+14335bg2+5993cg2\
   -3441dg2-11617eg2-13625fg2-15719g3
↦ j[111]=bd3g6-5519d2e2g6-1115be3g6+6336ce3g6-2069de3g6+2629e4g6+15344c3fg6\
   +6470bcdfg6-10902c2dfg6-1658bd2fg6+10724cd2fg6+14832d3fg6-13520bcefg6+962\
   3c2efg6+7786bdefg6-15791cdefg6-10914d2efg6+1690be2fg6+2498ce2fg6-2410de2f\
   g6+1627e3fg6-15385bcf2g6+7042c2f2g6-7786bdf2g6+13334cdf2g6-7629d2f2g6+149\
   05bef2g6-15392cef2g6-6534def2g6-13924e2f2g6+13562bf3g6+14423cf3g6+10846df\
   3g6+12131ef3g6-6187f4g6+3311c3g7-4717bcdg7+14916c2dg7+6338bd2g7-4934cd2g7\
   -13329d3g7-695bceg7-8327c2eg7+15569bdeg7-11427cdeg7-13441d2eg7-15017be2g7\
   -273ce2g7+11750de2g7-7341e3g7-11764bcfg7-7831c2fg7-10611bdfg7-7281cdfg7+3\
   925d2fg7-12870befg7+9363cefg7-15556defg7-14641e2fg7+10741bf2g7-2222cf2g7+\
   8884df2g7+1133ef2g7-13257f3g7-13206bcg8-11206c2g8+3434bdg8-13768cdg8+1038\
   1d2g8-3204beg8-11494ceg8-3629deg8+10685e2g8-8862bfg8+8588cfg8-10306dfg8+4\
   417efg8+12211f2g8-7933bg9+9250cg9+6382dg9+4064eg9+14713fg9+1352g10+12798c\
   3+11816bcd-2372c2d+8470bd2+9430cd2+6332d3+14288bce+13945c2e-2081bde-11849\
   cde-3576d2e-13041be2+14593ce2-12725de2+13481e3-12681bcf-8375c2f+7268bdf-3\
   649cdf+13505d2f+321bef-6341cef-6760def-3487e2f+1550bf2-1889cf2+7039df2-21\
   45ef2+12251f3-7535bcg-13639c2g+11649bdg+11287cdg-4180d2g-14051beg+3441ceg\
   +14440deg+12247e2g+12265bfg+15877cfg+13083dfg+8455efg+15228f2g-2394bg2+15\
   671cg2+13794dg2-14642eg2-1538fg2+9102g3
↦ j[112]=c4g6+13123d2e2g6-920be3g6+5453ce3g6-6616de3g6+11823e4g6+9895c3fg6-\
   7219bcdfg6-8607c2dfg6+1779bd2fg6-11714cd2fg6+2192d3fg6-15845bcefg6-11203c\
   2efg6+67bdefg6+10724cdefg6+10061d2efg6-8196be2fg6-6271ce2fg6+9476de2fg6-8\
   53e3fg6+10327bcf2g6-12783c2f2g6+5722bdf2g6-15750cdf2g6+3810d2f2g6+15606be\
   f2g6-8534cef2g6-12229def2g6-1286e2f2g6-3818bf3g6+13581cf3g6-7471df3g6-192\
   2ef3g6-6164f4g6+11513c3g7+217bcdg7-13041c2dg7-1948bd2g7+9318cd2g7-6822d3g\
   7-14136bceg7-12921c2eg7-961bdeg7-15312cdeg7+7872d2eg7-3170be2g7-14619ce2g\
   7-9886de2g7+2518e3g7+5913bcfg7-9689c2fg7+549bdfg7-13105cdfg7-9202d2fg7+15\
   297befg7-2997cefg7+4978defg7-4630e2fg7-950bf2g7-2061cf2g7+1169df2g7-365ef\
   2g7+6512f3g7-9204bcg8+12028c2g8+146bdg8+821cdg8-757d2g8-15555beg8+6783ceg\
   8-14648deg8+7508e2g8+7391bfg8+12189cfg8-8110dfg8-585efg8-14067f2g8-1768bg\
   9+3827cg9+921dg9-8135eg9+147fg9-1919g10-472c3+182bcd-13192c2d-7860bd2+291\
   9cd2-8369d3+8316bce+177c2e-4107bde-5041cde+13043d2e-9423be2-10786ce2-8800\
   de2-3070e3+3645bcf+12422c2f-12246bdf+7171cdf-1852d2f-4121bef+2111cef-6748\
   def-3887e2f-9913bf2-5170cf2+14115df2+10542ef2+11748f3+5783bcg+500c2g-2068\
   bdg-5732cdg-371d2g-8940beg+14944ceg+11085deg+5386e2g-12356bfg-10248cfg+89\
   82dfg-6885efg+5640f2g+9778bg2+1695cg2+74dg2-15956eg2+7741fg2+8309g3
↦ j[113]=f5g5+1303d2e2g6+14970be3g6+5805ce3g6-2426de3g6-5905e4g6+1402c3fg6+\
  5293bcdfg6+15561c2dfg6-10532bd2fg6+12164cd2fg6+670d3fg6-3835bcefg6-1580c2\
  efg6-15855bdefg6-6072cdefg6+1050d2efg6+7565be2fg6+80ce2fg6+381de2fg6+2488\
  e3fg6-13012bcf2g6+3012c2f2g6-13069bdf2g6-5163cdf2g6+596d2f2g6-8371bef2g6-\
  4886cef2g6+2708def2g6+12414e2f2g6-13268bf3g6+15388cf3g6+13169df3g6+3485ef\
  3g6-6229f4g6+8241c3g7+11996bcdg7-12865c2dg7-189bd2g7-756cd2g7-13873d3g7+1\
  1748bceg7-5650c2eg7-4920bdeg7-925cdeg7+11657d2eg7+6616be2g7+1276ce2g7+198\
  5de2g7+15637e3g7+10154bcfg7+2399c2fg7-11686bdfg7+11423cdfg7+6270d2fg7+517\
  0befg7+5754defg7-337e2fg7-13782bf2g7+15470cf2g7-15887df2g7+10684ef2g7-113\
  f3g7+7951bcg8+6900c2g8-223bdg8-2994cdg8-2954d2g8-4120beg8+13776ceg8-12256\
  deg8-4002e2g8-13238bfg8+3508cfg8+2953dfg8-5953efg8-2955f2g8+7152bg9+11664\
  cg9+9852dg9+6251eg9-554fg9-14474g10-11560c3-13244bcd+10072c2d-5851bd2-335\
  9cd2+856d3+14458bce-15027c2e+2864bde+15277cde-949d2e-10013be2+7334ce2-484\
  de2-8801e3+6227bcf-15793c2f-9029bdf+4336cdf-14907d2f+9592bef+9119cef-5762\
  def-6165e2f-12bf2-5502cf2+15868df2-6395ef2-11156f3+5001bcg-15966c2g+8690b\
  dg-8434cdg+6095d2g+3258beg-9010ceg+5385deg+1741e2g-11740bfg-2662cfg+1777d\
  fg+15152efg+975f2g+5281bg2+5750cg2+3471dg2+1775eg2+6399fg2-14012g3
↦ j[114]=ef4g5-9255d2e2g6-12750be3g6+5767ce3g6-5628de3g6+11998e4g6-3305c3fg\
  6+3637bcdfg6-2399c2dfg6+3428bd2fg6+1975cd2fg6+1198d3fg6+12769bcefg6+5871c\
  2efg6+8387bdefg6-13694cdefg6-8795d2efg6+66be2fg6+15882ce2fg6-2285de2fg6-1\
  4465e3fg6-11903bcf2g6+1689c2f2g6-15890bdf2g6+4954cdf2g6-13745d2f2g6+6536b\
  ef2g6-8412cef2g6+1924def2g6+13713e2f2g6-3074bf3g6+15608cf3g6+10766df3g6-7\
  988ef3g6-4042f4g6+7928c3g7-4643bcdg7-15612c2dg7-15499bd2g7-6152cd2g7+8986\
  d3g7+10972bceg7+3495c2eg7+4120bdeg7-11416cdeg7-14592d2eg7+6772be2g7+14708\
  ce2g7+9453de2g7+593e3g7-9494bcfg7+13003c2fg7+8842bdfg7-4674cdfg7-7366d2fg\
  7-14284befg7+3460cefg7-1715defg7+937e2fg7-8267bf2g7-3595cf2g7-5910df2g7-1\
  5495ef2g7-15883f3g7+13518bcg8+7581c2g8-14135bdg8-5611cdg8+15940d2g8-7822b\
  eg8-15302ceg8-62deg8+13910e2g8-12642bfg8+64cfg8+15803dfg8-2547efg8-15062f\
  2g8+12374bg9-1029cg9-13737dg9+818eg9+15113fg9-14941g10+14493c3+7775bcd+87\
  11c2d+6866bd2+9369cd2-4448d3-13220bce-5220c2e-9833bde+378cde+13051d2e-264\
  2be2+5046ce2-15959de2+9102e3+6595bcf-13771c2f+2871bdf-7907cdf+13852d2f+13\
  173bef+3273cef-2361def-5733e2f+11301bf2-3205cf2-517df2+4889ef2-10517f3+15\
  108bcg+8221c2g-929bdg+907cdg-2660d2g+14beg+10415ceg+9981deg+13973e2g+1262\
  1bfg+224cfg-6763dfg+3745efg-3216f2g-10718bg2+4956cg2+12172dg2-687eg2-4854\
  fg2+14639g3
↦ j[115]=df4g5-7211d2e2g6-7989be3g6+13706ce3g6+13184de3g6-8500e4g6+1391c3fg\
  6-5897bcdfg6+7682c2dfg6-2260bd2fg6-9442cd2fg6-12429d3fg6-11989bcefg6-1337\
  1c2efg6+6288bdefg6+7751cdefg6-12538d2efg6+11269be2fg6+7088ce2fg6+8646de2f\
  g6-13717e3fg6+557bcf2g6-471c2f2g6+8522bdf2g6-15516cdf2g6-13720d2f2g6-7273\
  bef2g6-12439cef2g6+5973def2g6-3875e2f2g6-4238bf3g6+6638cf3g6-11527df3g6-3\
  562ef3g6-10893f4g6-14629c3g7-1580bcdg7-13891c2dg7+15348bd2g7-9205cd2g7-14\
  277d3g7+12908bceg7-14431c2eg7+12749bdeg7+1549cdeg7+12267d2eg7+15540be2g7+\
  6728ce2g7+5056de2g7-3160e3g7-11221bcfg7+11419c2fg7+4605bdfg7-3130cdfg7+59\
  93d2fg7+7975befg7+13689cefg7-9517defg7+13166e2fg7+3364bf2g7-3480cf2g7-143\
  13df2g7+9149ef2g7+14712f3g7-8532bcg8+13937c2g8-2582bdg8+14000cdg8+3655d2g\
  8+2353beg8+11013ceg8-5856deg8-15262e2g8-13097bfg8+12542cfg8+12359dfg8-129\
  49efg8-2523f2g8-12384bg9-15198cg9-12309dg9-7282eg9+1550fg9-2579g10-9768c3\
  +13226bcd-13921c2d+11722bd2-10172cd2-7591d3-9297bce-7344c2e-10053bde+1495\
  4cde-9570d2e+11246be2+5934ce2-13403de2+5321e3-10318bcf+14312c2f-676bdf+12\
  445cdf-9277d2f+14315bef+6261cef+8916def-15708e2f-10170bf2-13372cf2+10402d\
  f2-10941ef2+6825f3-12511bcg+7413c2g+15624bdg+14964cdg+5651d2g+8313beg+791\
  4ceg-3580deg-15218e2g-11939bfg-3477cfg-8592dfg-4964efg+12942f2g+10085bg2+\
  11678cg2+15540dg2+13883eg2+11351fg2-5456g3
↦ j[116]=cf4g5+14890d2e2g6-8449be3g6-12484ce3g6-1303de3g6-10710e4g6-4213c3f\
   g6-14589bcdfg6-14413c2dfg6+9725bd2fg6+9692cd2fg6+11668d3fg6+11761bcefg6-1\
   1195c2efg6+8056bdefg6+6719cdefg6-11512d2efg6-762be2fg6+1117ce2fg6+13122de\
   2fg6+e3fg6+5634bcf2g6-13427c2f2g6-2234bdf2g6-529cdf2g6-8318d2f2g6-13529be\
   f2g6+7832cef2g6+9208def2g6+11418e2f2g6+4802bf3g6-9748cf3g6-14651df3g6+950\
   1ef3g6+13257f4g6-13251c3g7-2752bcdg7-15658c2dg7+15507bd2g7-9691cd2g7+1126\
   d3g7-12726bceg7+6734c2eg7-5889bdeg7-15139cdeg7-11957d2eg7+10742be2g7-1524\
   4ce2g7+4341de2g7+5000e3g7+10553bcfg7+10135c2fg7-2020bdfg7-1740cdfg7-11857\
   d2fg7+3993befg7+10411cefg7-11226defg7-4560e2fg7+3119bf2g7-15389cf2g7+4252\
   df2g7+3698ef2g7+6383f3g7-13551bcg8-7637c2g8+12744bdg8+9878cdg8-14530d2g8-\
   6081beg8-4742ceg8+8052deg8-9861e2g8-11391bfg8-8366cfg8-3985dfg8-5912efg8+\
   12659f2g8-12480bg9-3461cg9-10341dg9-1491eg9-10461fg9-4247g10-5826c3+15477\
   bcd-13111c2d-3830bd2+1084cd2-9870d3+10649bce+13526c2e-41bde-2479cde+997d2\
   e-5978be2+5843ce2+14997de2+11490e3-14544bcf+2299c2f-11807bdf+2184cdf+2672\
   d2f+8381bef+5574cef+5737def-12257e2f+13791bf2+7651cf2+10376df2-12291ef2+1\
   0028f3-8385bcg+1120c2g-13248bdg-13354cdg-12971d2g-6022beg-2929ceg+11930de\
   g-4237e2g+1329bfg+789cfg+6529dfg-14675efg-935f2g-7958bg2-6221cg2+8415dg2-\
   3556eg2-13707fg2+9320g3
↦ j[117]=bf4g5-872d2e2g6+2360be3g6-7674ce3g6+13232de3g6-3451e4g6-424c3fg6-1\
   2359bcdfg6+6567c2dfg6+6213bd2fg6+2590cd2fg6+14611d3fg6-2626bcefg6-3822c2e\
   fg6+6519bdefg6-10659cdefg6+10481d2efg6-6698be2fg6-2493ce2fg6-8015de2fg6+9\
   884e3fg6+1206bcf2g6-11250c2f2g6-2318bdf2g6-5567cdf2g6-6892d2f2g6-6344bef2\
   g6+79cef2g6-1285def2g6+4898e2f2g6+61bf3g6+11349cf3g6-1354df3g6+4606ef3g6-\
   5264f4g6-4675c3g7+15880bcdg7+6997c2dg7-3571bd2g7-13506cd2g7-11465d3g7-159\
   39bceg7+12641c2eg7-4614bdeg7-5230cdeg7-14820d2eg7+13987be2g7+12415ce2g7+1\
   4487de2g7-9608e3g7+9791bcfg7-10977c2fg7+11908bdfg7+11827cdfg7-2677d2fg7-5\
   334befg7+8626cefg7-2162defg7-8298e2fg7+1953bf2g7+1329cf2g7+12552df2g7+461\
   3ef2g7-5688f3g7-5664bcg8-827c2g8-4976bdg8+2462cdg8+6326d2g8-14216beg8-179\
   ceg8+10530deg8+6988e2g8-7718bfg8+15078cfg8-6352dfg8-3998efg8-10388f2g8+10\
   766bg9+10679cg9-15843dg9+2286eg9+8148fg9+5788g10-15608c3-13968bcd+4113c2d\
   +7071bd2+4527cd2+3633d3+5499bce+2613c2e+10701bde+14757cde+13232d2e-3375be\
   2-14408ce2-14558de2+13626e3+429bcf+6648c2f-8640bdf-1755cdf+5208d2f+3117be\
   f-2046cef-7024def+8262e2f+14397bf2+5424cf2-5942df2+5158ef2+673f3-14925bcg\
   -195c2g+12457bdg-15207cdg-10494d2g+900beg-13799ceg+7881deg+14292e2g+1515b\
   fg-11769cfg+1589dfg-8084efg-2151f2g-3446bg2-12723cg2+3215dg2+10286eg2-663\
   fg2+6938g3
↦ j[118]=e2f3g5-421d2e2g6+1230be3g6+4014ce3g6+8796de3g6+4958e4g6+741c3fg6+4\
   017bcdfg6-10254c2dfg6-4430bd2fg6+5622cd2fg6+7207d3fg6-9588bcefg6-9224c2ef\
   g6+1835bdefg6+6665cdefg6-4819d2efg6-5392be2fg6-7435ce2fg6+8640de2fg6-1561\
   6e3fg6+15363bcf2g6-2501c2f2g6+605bdf2g6-10329cdf2g6-7911d2f2g6+7712bef2g6\
   -6988cef2g6-15223def2g6+327e2f2g6-4042bf3g6-11394cf3g6-293df3g6-3750ef3g6\
   +8753f4g6-11288c3g7+13645bcdg7-9447c2dg7-2854bd2g7+14014cd2g7-3297d3g7-15\
   821bceg7+12823c2eg7+14558bdeg7-8797cdeg7-3412d2eg7+2028be2g7-15681ce2g7+1\
   5292de2g7+3317e3g7-9351bcfg7-13930c2fg7-9672bdfg7+7835cdfg7-3977d2fg7-490\
   3befg7+14479cefg7+12201defg7-5375e2fg7+13140bf2g7+1103cf2g7-9508df2g7-211\
   9ef2g7-8166f3g7-1371bcg8+11272c2g8-1555bdg8+6296cdg8-10804d2g8-2685beg8-6\
   899ceg8-1024deg8-11339e2g8+12491bfg8-13792cfg8-5791dfg8+3429efg8-12121f2g\
   8-2142bg9+14836cg9-8155dg9+10053eg9+5078fg9+15733g10+9306c3-12017bcd-4668\
   c2d-15759bd2-4377cd2+7801d3+3239bce+15244c2e-9022bde+4438cde+10225d2e-202\
   be2+8968ce2+13144de2+148e3+13702bcf-9063c2f-12368bdf+11714cdf-9621d2f-124\
   95bef+229cef-15059def-5561e2f+14591bf2-11089cf2+9829df2-6597ef2-10762f3-9\
   988bcg+6614c2g-5893bdg+9978cdg-8681d2g+10258beg-8873ceg-13794deg+1859e2g-\
   4650bfg+9954cfg+9083dfg-11999efg-1535f2g-4217bg2-10204cg2+9888dg2+93eg2-1\
   2424fg2+9381g3
```

```
↦ j[119]=def3g5+6034d2e2g6-5377be3g6+6547ce3g6-11350de3g6+11954e4g6-2402c3f\
  g6+1536bcdfg6+303c2dfg6-970bd2fg6+996cd2fg6-5378d3fg6+4965bcefg6-15565c2e\
  fg6-80bdefg6-776cdefg6-13776d2efg6+10088be2fg6-13747ce2fg6-6944de2fg6-122\
  4e3fg6-9785bcf2g6+2642c2f2g6+2394bdf2g6-6187cdf2g6-12024d2f2g6-4917bef2g6\
  -14583cef2g6+2530def2g6-1227e2f2g6-2176bf3g6+1033cf3g6-4535df3g6+12189ef3\
  g6-7514f4g6-6920c3g7+3724bcdg7-1428c2dg7+5748bd2g7-6906cd2g7+3017d3g7+892\
  9bceg7+9101c2eg7-15541bdeg7-4540cdeg7-3610d2eg7-10698be2g7-15664ce2g7-118\
  66de2g7+13543e3g7+13221bcfg7+116c2fg7-3673bdfg7+10192cdfg7+2220d2fg7+1063\
  6befg7-9989cefg7+14662defg7-5977e2fg7+10119bf2g7-1624cf2g7-10286df2g7+710\
  0ef2g7+5098f3g7+3762bcg8-3990c2g8-14190bdg8-14646cdg8-11650d2g8-14412beg8\
  -13548ceg8-14983deg8+5387e2g8+6152bfg8+13757cfg8+337dfg8+2398efg8+15005f2\
  g8+8565bg9-3202cg9+7019dg9-11905eg9-1376fg9-11968g10+4009c3-15202bcd+8842\
  c2d-4988bd2-8860cd2+15626d3+14875bce+12202c2e-8694bde+14728cde-1960d2e+11\
  421be2+815ce2+15315de2+3310e3-2314bcf+3941c2f+400bdf+12821cdf-13422d2f+10\
  193bef-6617cef-12985def+8499e2f+9938bf2+9270cf2+4081df2-11767ef2-11722f3-\
  3324bcg+14974c2g+11884bdg+3505cdg+868d2g+15247beg-4892ceg-298deg+308e2g+3\
  522bfg+13519cfg-6202dfg-10273efg-11070f2g+13028bg2+13386cg2-6280dg2-4832e\
  g2-6953fg2-7766g3
↦ j[120]=cef3g5-6537d2e2g6-7888be3g6+12464ce3g6-3103de3g6-2810e4g6-3175c3fg\
  6+9990bcdfg6+8320c2dfg6-1772bd2fg6-7966cd2fg6+7471d3fg6-966bcefg6-252c2ef\
  g6+13251bdefg6+12087cdefg6+6103d2efg6+3973be2fg6+15851ce2fg6-1039de2fg6-9\
  496e3fg6-14055bcf2g6-4970c2f2g6-2815bdf2g6-10950cdf2g6-3690d2f2g6-13106be\
  f2g6+1652cef2g6-13537def2g6+13752e2f2g6-24bf3g6+8402cf3g6-15388df3g6-9560\
  ef3g6+5396f4g6+13382c3g7-8986bcdg7-14963c2dg7-12793bd2g7+8320cd2g7-6255d3\
  g7-14633bceg7-4370c2eg7-13630bdeg7-8775cdeg7-10293d2eg7+56be2g7-2262ce2g7\
  -3718de2g7-13208e3g7-3176bcfg7+9c2fg7-11134bdfg7-1038cdfg7+13776d2fg7+101\
  85befg7+4606cefg7+3268defg7-1614e2fg7+13193bf2g7-15398cf2g7+3540df2g7+120\
  0ef2g7+14189f3g7+1008bcg8+10096c2g8-11449bdg8-7015cdg8-9247d2g8+10207beg8\
  -13593ceg8+8233deg8-15481e2g8-14793bfg8-13438cfg8-1378dfg8-5973efg8+1867f\
  2g8-11588bg9+14261cg9-7997dg9+14831eg9+14252fg9+3047g10-5694c3-149bcd-429\
  0c2d-11454bd2+15329cd2+12137d3+12476bce-542c2e-3641bde+11574cde+5882d2e-9\
  364be2-2728ce2+4118de2+9647e3-2548bcf-9689c2f-1858bdf+5649cdf-10598d2f+14\
  942bef+1441cef-11936def+8126e2f-9852bf2+4066cf2-7572df2+8482ef2-14506f3+3\
  655bcg-10684c2g-409bdg+12986cdg+5122d2g+8895beg-4315ceg-5045deg-7620e2g-8\
  798bfg-261cfg-5773dfg+6126efg+832f2g-6959bg2-526cg2-9254dg2+6310eg2+15771\
  fg2+5548g3
↦ j[121]=bef3g5-11837d2e2g6-14441be3g6+5998ce3g6+10898de3g6-8786e4g6-4659c3\
  fg6-11397bcdfg6+1497c2dfg6-2328bd2fg6+2064cd2fg6-6620d3fg6-3311bcefg6-228\
  c2efg6+15716bdefg6+5688cdefg6+12620d2efg6+2653be2fg6-1426ce2fg6+15885de2f\
  g6+52e3fg6-8420bcf2g6+3798c2f2g6+8313bdf2g6+978cdf2g6+1522d2f2g6+4806bef2\
  g6-631cef2g6-9132def2g6+7841e2f2g6-12922bf3g6-6831cf3g6-141df3g6-7173ef3g\
  6-15691f4g6-8081c3g7+15735bcdg7-11199c2dg7-12324bd2g7-2978cd2g7+12456d3g7\
  -10968bceg7+1882c2eg7+3914bdeg7+11056cdeg7-11218d2eg7+2066be2g7-4706ce2g7\
  -3276de2g7+8280e3g7+692bcfg7+11247c2fg7-4494bdfg7-6645cdfg7-1542d2fg7-100\
  50befg7-13168cefg7+13891defg7-51e2fg7+12521bf2g7+1158cf2g7+13739df2g7+533\
  5ef2g7-5016f3g7+5392bcg8+10234c2g8+4499bdg8+2166cdg8+1831d2g8-7069beg8+75\
  80ceg8+11924deg8-5865e2g8+7263bfg8+356cfg8-9120dfg8+1788efg8+12657f2g8-10\
  395bg9-9327cg9+13890dg9+13316eg9+4704fg9-891g10-12175c3+12885bcd-6028c2d-\
  4652bd2+6405cd2-1919d3+15688bce-754c2e-3156bde-11829cde-15387d2e-14430be2\
  -9374ce2-14211de2+5230e3+375bcf-8464c2f-15310bdf+15690cdf-8527d2f-4814bef\
  +355cef-1497def+14240e2f+9481bf2+2813cf2+4006df2-14643ef2-1752f3+7352bcg-\
  13616c2g+2734bdg+9229cdg+11169d2g+11468beg+14293ceg-11964deg+876e2g+351bf\
  g+6523cfg+8921dfg-1009efg-8827f2g+6576bg2+10300cg2+9728dg2-6142eg2+10878f\
   g2-12628g3
↦ j[122]=d2f3g5+13897d2e2g6+4299be3g6-14765ce3g6-1275de3g6+6604e4g6+7658c3f\
   g6+10769bcdfg6+794c2dfg6+12176bd2fg6+15940cd2fg6+7784d3fg6-10780bcefg6-97\
   00c2efg6+782bdefg6+2296cdefg6+14121d2efg6-6353be2fg6-7067ce2fg6+6221de2fg\
   6-1415e3fg6-7331bcf2g6-12085c2f2g6+5244bdf2g6-5055cdf2g6-13047d2f2g6-1341\
   0bef2g6-7577cef2g6+4207def2g6+7951e2f2g6+2846bf3g6+3470cf3g6-4021df3g6-18\
   62ef3g6+5900f4g6-3933c3g7-933bcdg7+9064c2dg7-6054bd2g7-12458cd2g7+10791d3\
   g7-7515bceg7+3705c2eg7-3632bdeg7-11357cdeg7-6579d2eg7+13407be2g7-4686ce2g\
   7+8033de2g7-9645e3g7+13032bcfg7-4808c2fg7-9706bdfg7+13639cdfg7+3633d2fg7+\
   7304befg7+13044cefg7+8947defg7+6404e2fg7-15143bf2g7-1745cf2g7+5013df2g7-3\
   238ef2g7+9165f3g7+9171bcg8-4423c2g8-2396bdg8+10113cdg8-4005d2g8+15103beg8\
   -9251ceg8+9480deg8-7792e2g8-6330bfg8+6280cfg8+11780dfg8+6345efg8-5385f2g8\
   +669bg9-6474cg9-12921dg9-6753eg9+4530fg9-11300g10-2781c3+15517bcd+13417c2\
   d-13605bd2+7611cd2+9392d3-9116bce-11820c2e+3267bde+11027cde+12552d2e+1349\
   be2-350ce2-13641de2+1563e3+4125bcf-2805c2f+7971bdf+12567cdf+14319d2f-1088\
   4bef+527cef-7256def-861e2f-14690bf2+9396cf2-5847df2-292ef2+2027f3-7370bcg\
   +3c2g-2197bdg-11284cdg-15096d2g-2050beg-9052ceg-13912deg-10309e2g-11886bf\
   g+14232cfg+4765dfg-10131efg-5683f2g+2347bg2+15479cg2+5667dg2+3444eg2-1412\
   7fg2-2926g3
↦ j[123]=cdf3g5+9304d2e2g6+8650be3g6-15250ce3g6+13233de3g6+15726e4g6+7491c3\
   fg6+2216bcdfg6+13992c2dfg6+13166bd2fg6-14664cd2fg6+4569d3fg6-3201bcefg6-2\
   645c2efg6+4718bdefg6-8811cdefg6-1007d2efg6-13388be2fg6-375ce2fg6+9736de2f\
   g6-8550e3fg6-4413bcf2g6+7230c2f2g6-7319bdf2g6-6919cdf2g6-11672d2f2g6+5281\
   bef2g6+2904cef2g6-160def2g6+7243e2f2g6-3936bf3g6-15807cf3g6+13787df3g6+34\
   0ef3g6-7376f4g6-3732c3g7-5483bcdg7-11248c2dg7-3783bd2g7-14836cd2g7+4106d3\
   g7+2772bceg7+11596c2eg7-13332bdeg7-13932cdeg7-6769d2eg7+744be2g7-12405ce2\
   g7+24de2g7+11045e3g7-71bcfg7-12842c2fg7+13975bdfg7-6316cdfg7-15309d2fg7+1\
   0839befg7-15841cefg7+14617defg7-10010e2fg7+42bf2g7-14430cf2g7+9719df2g7-2\
   654ef2g7+4269f3g7-12274bcg8-7362c2g8-12367bdg8+14634cdg8+2565d2g8+1784beg\
   8-13954ceg8+14848deg8-5278e2g8+4368bfg8+11764cfg8+1178dfg8+11193efg8+9420\
   f2g8-8079bg9-8976cg9+4256dg9+15525eg9-14541fg9+10552g10+1808c3+8912bcd-11\
   655c2d-9985bd2-13996cd2-15439d3+4789bce-1517c2e+9801bde+3508cde+13190d2e+\
   3688be2+7433ce2-11741de2-1814e3+11143bcf+2525c2f+3933bdf+9186cdf-5209d2f+\
   14995bef-9861cef-9209def+11728e2f+3089bf2+12560cf2-14438df2-7592ef2+12725\
   f3-8271bcg-1690c2g+2556bdg-13137cdg+9255d2g-4347beg+12324ceg+1247deg-1892\
   e2g+5942bfg-15427cfg-774dfg-2000efg+3377f2g+4849bg2+6822cg2+4433dg2-8256e\
   g2-9507fg2+5831g3
↦ j[124]=bdf3g5-14560d2e2g6-9156be3g6-1387ce3g6+11788de3g6-4923e4g6+7701c3f\
   g6+4565bcdfg6-11358c2dfg6+5148bd2fg6-11059cd2fg6-3916d3fg6-12171bcefg6+98\
   37c2efg6+13429bdefg6+15057cdefg6+3882d2efg6+5298be2fg6+2727ce2fg6+9746de2\
   fg6-5681e3fg6+6556bcf2g6+10051c2f2g6+2837bdf2g6+8973cdf2g6-1684d2f2g6-920\
   8bef2g6+12036cef2g6-656def2g6-2656e2f2g6+4576bf3g6+2086cf3g6+2387df3g6+71\
   36ef3g6+13723f4g6+4430c3g7+8732bcdg7+4766c2dg7+4930bd2g7+12397cd2g7-177d3\
   g7+7040bceg7-4384c2eg7+12642bdeg7-3014cdeg7+13295d2eg7-11685be2g7-12844ce\
   2g7+10263de2g7+6052e3g7-5671bcfg7+9726c2fg7-10244bdfg7-4051cdfg7+11659d2f\
   g7+11330befg7-4615cefg7+6072defg7+11125e2fg7+8325bf2g7+10393cf2g7-7581df2\
   g7-1756ef2g7-2598f3g7-14139bcg8+356c2g8-99bdg8+3655cdg8+13387d2g8-533beg8\
   -15315ceg8-10959deg8-1865e2g8+4082bfg8-6384cfg8-7930dfg8-11789efg8+15544f\
   2g8-2327bg9+5252cg9-4249dg9-14340eg9-14017fg9-760g10-5051c3+2990bcd-14011\
   c2d+5411bd2+7841cd2-14301d3+5485bce+1077c2e-3257bde+5456cde-10134d2e+58be\
   2-7122ce2+9281de2+12486e3-13225bcf+1975c2f-13004bdf+15141cdf+3759d2f-5618\
   bef-228cef-2530def-5051e2f+1423bf2+13591cf2-12820df2-11184ef2-10655f3+135\
   26bcg-8030c2g+14772bdg-14314cdg+11024d2g-9918beg-5041ceg-1729deg+4132e2g-\
   2192bfg-7427cfg+11771dfg-411efg-4152f2g+11294bg2-3369cg2+8902dg2-11394eg2\
   -12730fg2-13746g3
↦ j[125]=c2f3g5+14199d2e2g6+6910be3g6-11092ce3g6+472de3g6-15338e4g6-1809c3f\
   g6-4673bcdfg6-9475c2dfg6+9272bd2fg6-7651cd2fg6-1445d3fg6-10773bcefg6+1591\
   9c2efg6+4734bdefg6+4754cdefg6+6427d2efg6-3098be2fg6-9682ce2fg6-12074de2fg\
   6+7585e3fg6-312bcf2g6+9373c2f2g6+3520bdf2g6-4635cdf2g6-6105d2f2g6-907bef2\
   g6-4331cef2g6+7529def2g6+13651e2f2g6+15993bf3g6+12130cf3g6+7341df3g6-1281\
   3ef3g6+7738f4g6+10071c3g7+15879bcdg7+3399c2dg7+9559bd2g7+11466cd2g7+513d3\
   g7-11824bceg7-227c2eg7+15734bdeg7+1030cdeg7+5615d2eg7+10234be2g7+457ce2g7\
   -14239de2g7-8867e3g7-6200bcfg7+142c2fg7+1768bdfg7-1872cdfg7+9507d2fg7-333\
   0befg7-11557cefg7+3634defg7-9159e2fg7+2798bf2g7+4414cf2g7-2878df2g7-2777e\
   f2g7-11284f3g7-789bcg8+3084c2g8+1663bdg8-9321cdg8+15945d2g8+6317beg8+8317\
   ceg8+13581deg8+7651e2g8+10384bfg8+12623cfg8-3171dfg8-5543efg8+12820f2g8-6\
   585bg9+4646cg9-714dg9+2089eg9-3034fg9-9705g10-11864c3+9178bcd-11489c2d-41\
   8bd2-11151cd2-2477d3-12925bce+5026c2e-15911bde-8873cde-12431d2e+4592be2+3\
   96ce2-9345de2-9802e3-11638bcf-7845c2f-3530bdf-14988cdf+3644d2f-15752bef+4\
   966cef+9324def-5474e2f+6474bf2-13436cf2-9808df2-10837ef2-3428f3+7361bcg+1\
   4974c2g+2556bdg-329cdg-7498d2g-5516beg-14153ceg-3586deg+3138e2g-12772bfg-\
   2219cfg-1929dfg-7177efg+15015f2g+14030bg2-362cg2-7167dg2+8278eg2+12514fg2\
   -938g3
↦ j[126]=bcf3g5+6820d2e2g6+5187be3g6-4585ce3g6+518de3g6+7291e4g6-11308c3fg6\
   +3958bcdfg6+6530c2dfg6+2942bd2fg6+7417cd2fg6-9059d3fg6+4867bcefg6-2454c2e\
   fg6-10772bdefg6-4967cdefg6+4622d2efg6+11542be2fg6-7524ce2fg6-92de2fg6+308\
   6e3fg6+7266bcf2g6-15084c2f2g6-1145bdf2g6-10961cdf2g6-4590d2f2g6-2316bef2g\
   6-9717cef2g6+7441def2g6+15816e2f2g6-213bf3g6-5606cf3g6+10285df3g6-6009ef3\
   g6+2896f4g6+6038c3g7+5132bcdg7+9284c2dg7+10660bd2g7-13382cd2g7+2252d3g7+3\
   908bceg7-768c2eg7+9017bdeg7+6445cdeg7+12527d2eg7+14849be2g7+2842ce2g7+515\
   9de2g7+9193e3g7-1402bcfg7-13099c2fg7-15035bdfg7+9450cdfg7+13318d2fg7-1151\
   8befg7+1303cefg7+8675defg7-9235e2fg7-12694bf2g7-678cf2g7+10335df2g7+5560e\
   f2g7+6354f3g7+1874bcg8-11102c2g8+12855bdg8-5545cdg8+8619d2g8+14898beg8-15\
   780ceg8-4268deg8-3540e2g8-6006bfg8+2900cfg8+14955dfg8+12997efg8-3715f2g8+\
   6921bg9+11999cg9+15527dg9-5978eg9-15796fg9+1892g10-7949c3+1145bcd+6281c2d\
   -11884bd2+11009cd2+4359d3-7199bce-11215c2e-9769bde-880cde-15422d2e+12668b\
   e2+376ce2+10453de2+2767e3+1864bcf-14238c2f+7359bdf-2754cdf-15221d2f+15712\
   bef+6841cef+8526def-12722e2f+953bf2-1868cf2+8649df2-12708ef2-14967f3+1848\
   bcg-15752c2g+12591bdg-214cdg+9948d2g-11072beg-6236ceg-5578deg+6670e2g-390\
   8bfg-8621cfg-4165dfg+2632efg-1832f2g-1078bg2-7358cg2+8581dg2-13219eg2+133\
   12fg2-14992g3
↦ j[127]=e3f2g5-1796d2e2g6+11352be3g6-3580ce3g6+5996de3g6-5103e4g6-6965c3fg\
   6+3905bcdfg6+8283c2dfg6+700bd2fg6+14770cd2fg6-13314d3fg6-4300bcefg6+6626c\
   2efg6-14379bdefg6+15897cdefg6+11609d2efg6-806be2fg6-2340ce2fg6+6903de2fg6\
   +10799e3fg6-8321bcf2g6+192c2f2g6+943bdf2g6-5783cdf2g6+5359d2f2g6+4120bef2\
   g6-11803cef2g6-315def2g6+1851e2f2g6-1693bf3g6-14999cf3g6-13198df3g6-10221\
   ef3g6-8581f4g6+1404c3g7+4120bcdg7-10616c2dg7-8041bd2g7+4059cd2g7-9557d3g7\
   -76bceg7-10400c2eg7+438bdeg7+8911cdeg7-7936d2eg7+13361be2g7+2752ce2g7-816\
   1de2g7+10214e3g7-13491bcfg7+14992c2fg7+1343bdfg7+1104cdfg7-14732d2fg7+485\
   0befg7-14011cefg7+5759defg7+3112e2fg7-12690bf2g7-12085cf2g7-15618df2g7-45\
   46ef2g7-5270f3g7+3079bcg8-927c2g8+7312bdg8-5887cdg8-2847d2g8+15817beg8+12\
   609ceg8+13662deg8-13174e2g8+4275bfg8+5621cfg8+4872dfg8-507efg8-3787f2g8-1\
   4790bg9-11080cg9+3041dg9-5002eg9-901fg9-1083g10-7235c3-13139bcd-10407c2d+\
   13361bd2+13403cd2-10479d3+3168bce-10379c2e-13448bde+7747cde+14290d2e-1551\
   5be2-3621ce2+9339de2+12827e3-13964bcf+11096c2f-13948bdf-8972cdf-2872d2f-6\
   934bef+5952cef+6591def+8724e2f+7798bf2-9537cf2-15741df2+839ef2+4203f3+105\
   47bcg-14110c2g+2223bdg-12537cdg+11278d2g-11126beg+6835ceg+12991deg-9765e2\
   g+491bfg+972cfg-4910dfg+6548efg+10195f2g-10548bg2-15098cg2+6228dg2-4582eg\
   2-14797fg2-15298g3
↦ j[128]=de2g8-3906e3g8+11290bcfg8-5212c2fg8-12729bdfg8-3843cdfg8-9429d2fg8\
   +5327befg8-12702cefg8+9832defg8-9658e2fg8+8454bf2g8+15955cf2g8-14224df2g8\
   +11462ef2g8+7488f3g8+7264bcg9+10370c2g9+1398bdg9+540cdg9+2597d2g9+8268beg\
   9-11891ceg9-5031deg9+7694e2g9-3427bfg9+9333cfg9-13787dfg9+12181efg9+3762f\
   2g9-2537bg10+12607cg10-10540dg10-429eg10-4100fg10+3744g11-15544c4+5453bd3\
   -9677cd3+5773d4+1872c3e+9167bcde+15668c2de-2905bd2e+4512cd2e-9323d3e-4667\
   bce2-1978c2e2-10381bde2+1586cde2+12567d2e2+14005be3-3026ce3+5811de3+3995e\
   4-9554c3f+11186bcdf-7465c2df-6799bd2f+9079cd2f-11867d3f+357bcef+1667c2ef+\
   12673bdef+14340cdef-5748d2ef-4074be2f+12479ce2f-15227de2f+7483e3f+1376bcf\
   2-8512c2f2-15233bdf2-15616cdf2+13178d2f2-13611bef2+4870cef2+4936def2+1521\
   8e2f2+15459bf3-11073cf3+15921df3+10125ef3-10278f4+8785c3g-7944bcdg-886c2d\
   g+1550bd2g-2489cd2g+14811d3g-1179bceg-9413c2eg+11308bdeg-809cdeg+11453d2e\
   g-7283be2g+12450ce2g-8113de2g+12011e3g-7857bcfg+1120c2fg-5861bdfg+15439cd\
   fg+4655d2fg+7935befg-6230cefg-9409defg-2169e2fg+4429bf2g+13922cf2g-2661df\
   2g+4152ef2g-4028f3g-3298bcg2+15598c2g2+10868bdg2-12120cdg2-9520d2g2-10077\
   beg2-6858ceg2-12994deg2-7784e2g2-8493bfg2+362cfg2+11303dfg2+8620efg2-9078\
   f2g2+12591bg3-13914cg3-755dg3-14035eg3-9161fg3+10742g4
↦ j[129]=ce2g8+10163e3g8+4152bcfg8+3803c2fg8-3304bdfg8+959cdfg8+10360d2fg8-\
   14462befg8+2604cefg8+11337defg8+13998e2fg8-6769bf2g8-15187cf2g8+14846df2g\
   8+8718ef2g8-11103f3g8-912bcg9+12503c2g9-6212bdg9+2813cdg9-12550d2g9-15923\
   beg9+5531ceg9+1531deg9+3706e2g9+13537bfg9-8457cfg9+13417dfg9-627efg9+607f\
   2g9-10104bg10+5579cg10+199dg10+10816eg10-3586fg10-11626g11+5671c4-3004bd3\
   +6491cd3-4383d4+14997c3e+3553bcde-14631c2de+1590bd2e-6516cd2e+15229d3e+12\
   801bce2+6311c2e2-11423bde2-12906cde2+13009d2e2-9815be3-10784ce3+13368de3+\
   10857e4+6331c3f-2495bcdf-12768c2df-498bd2f-10714cd2f+9293d3f-13448bcef-88\
   2c2ef-11121bdef+12101cdef+7274d2ef+4700be2f-7879ce2f-842de2f-13708e3f+991\
   3bcf2+4227c2f2+5428bdf2+12959cdf2+15897d2f2-10208bef2-1413cef2-15399def2-\
   12286e2f2+1905bf3-5988cf3+13933df3+5262ef3+4963f4-15979c3g+8980bcdg+7178c\
   2dg+14158bd2g-11759cd2g+10105d3g+7350bceg+13332c2eg-3325bdeg+8801cdeg+32d\
   2eg-3741be2g-13186ce2g+10465de2g-14904e3g+2488bcfg+7543c2fg-9556bdfg+1059\
   3cdfg+8620d2fg-13287befg-7903cefg+5231defg-3659e2fg+2726bf2g+6843cf2g-150\
   42df2g+12081ef2g-8140f3g+1258bcg2+11009c2g2-11671bdg2+12573cdg2-3357d2g2-\
   15574beg2-4338ceg2-9509deg2+783e2g2-14040bfg2-3492cfg2-513dfg2-1815efg2+5\
   841f2g2+344bg3+6882cg3+10627dg3+11560eg3+903fg3-4820g4
↦ j[130]=be2g8+9955e3g8-1388bcfg8+6753c2fg8+5625bdfg8+14147cdfg8+8324d2fg8-\
   3459befg8-12978cefg8+11256defg8-6738e2fg8-5535bf2g8-3823cf2g8+7120df2g8+1\
   124ef2g8-5154f3g8+3306bcg9+8244c2g9+6177bdg9+2930cdg9+7087d2g9+10115beg9-\
   7538ceg9-1559deg9-15681e2g9+2068bfg9-14455cfg9-6499dfg9+5582efg9-1665f2g9\
   +2779bg10-4590cg10+9645dg10-7150eg10+9856fg10+13886g11-2438c4+14315bd3+12\
   955cd3-8049d4+5316c3e+13335bcde-10037c2de+6127bd2e-11948cd2e+6662d3e+183b\
   ce2+6793c2e2-11878bde2-3323cde2-8652d2e2-6801be3-15596ce3-10478de3+9056e4\
   -12824c3f+15925bcdf+48c2df-6331bd2f-8893cd2f+5324d3f+6506bcef-12633c2ef-7\
   736bdef+7120cdef-4229d2ef+14349be2f+8984ce2f-4732de2f-7909e3f+7350bcf2+14\
   823c2f2+9215bdf2-4391cdf2+2606d2f2-1505bef2+10521cef2-6400def2+6147e2f2-1\
   139bf3+1971cf3+6789df3+1596ef3+2121f4+7047c3g+13773bcdg+3863c2dg-9281bd2g\
   -7717cd2g+3669d3g+11395bceg-11247c2eg+9946bdeg+10688cdeg+5267d2eg-9060be2\
   g+9855ce2g-299de2g-244e3g+366bcfg-10016c2fg+4930bdfg-3564cdfg-7823d2fg+12\
   550befg-12809cefg-13009defg+6243e2fg-13088bf2g-14318cf2g+3730df2g+14386ef\
   2g+14757f3g+9643bcg2+2707c2g2-4488bdg2+7795cdg2-4149d2g2+2788beg2-11551ce\
   g2-15149deg2-11393e2g2-999bfg2-7461cfg2+15120dfg2+4237efg2-12090f2g2-4070\
   bg3-7194cg3-13607dg3+9423eg3-9598fg3-13934g4
↦ j[131]=d2eg8+8063e3g8+14280bcfg8+15811c2fg8+12244bdfg8-12867cdfg8-5837d2f\
   g8-3747befg8+11294cefg8-909defg8+10840e2fg8+10820bf2g8-9748cf2g8-10839df2\
   g8-13985ef2g8+3786f3g8+8585bcg9+8149c2g9-4041bdg9-10568cdg9-12894d2g9-139\
   67beg9-12246ceg9+11305deg9-15842e2g9-1388bfg9+345cfg9+15245dfg9-12330efg9\
   +6706f2g9-4739bg10-3729cg10+11724dg10-3821eg10-8514fg10+1919g11+1064c4+14\
   735bd3+478cd3+9813d4-1825c3e+15947bcde+12134c2de-4336bd2e+10569cd2e+9373d\
   3e-2093bce2-13602c2e2+2470bde2-10120cde2-6778d2e2+5147be3+254ce3+8772de3+\
   10554e4-12374c3f-5842bcdf+4985c2df-12304bd2f+13191cd2f-1599d3f+349bcef+10\
   730c2ef-10877bdef+10959cdef-4534d2ef-3031be2f-12992ce2f-7061de2f+1324e3f-\
   6446bcf2-11760c2f2+3481bdf2+12670cdf2+12809d2f2+8086bef2-9624cef2-580def2\
   -9262e2f2-11690bf3+252cf3-15128df3+15709ef3-11250f4-12682c3g-12152bcdg+38\
   83c2dg-1122bd2g-12474cd2g+12655d3g+12473bceg+1488c2eg+12011bdeg-14114cdeg\
   +12957d2eg-7260be2g+8665ce2g+1355de2g-13277e3g+12697bcfg-5502c2fg-8909bdf\
   g-1208cdfg+2581d2fg+11708befg-4996cefg+14346defg+12282e2fg-14760bf2g+7512\
   cf2g+11327df2g-9342ef2g+1023f3g-11437bcg2-747c2g2-13965bdg2+14918cdg2-859\
   4d2g2+12645beg2-3395ceg2-3163deg2-14576e2g2+4947bfg2-11994cfg2+15375dfg2+\
   14443efg2+2238f2g2-8519bg3+11872cg3-6171dg3+11741eg3+8621fg3-3254g4
↦ j[132]=cdeg8+9878e3g8-15593bcfg8+5661c2fg8-7825bdfg8+771cdfg8-5581d2fg8-1\
   5117befg8-11903cefg8-10552defg8+9930e2fg8+14001bf2g8-5017cf2g8+3732df2g8-\
   15621ef2g8+15328f3g8-356bcg9+10066c2g9-5639bdg9+15467cdg9+1136d2g9-7739be\
   g9+10641ceg9-11498deg9+2562e2g9+6946bfg9+10318cfg9+1854dfg9+9761efg9+5614\
   f2g9-6326bg10-4680cg10+5847dg10+14882eg10+2518fg10-4646g11+8690c4-141bd3-\
   2164cd3+3645d4+4247c3e-5755bcde-1552c2de-12359bd2e+5611cd2e+4918d3e+11922\
   bce2+8410c2e2-8208bde2-9905cde2+4298d2e2-11704be3+1589ce3-7172de3-9790e4-\
   11179c3f-2594bcdf+11091c2df-6839bd2f-5533cd2f+8895d3f+9436bcef+7211c2ef+8\
   429bdef+2439cdef-4463d2ef+13124be2f-9726ce2f-11837de2f-344e3f+1481bcf2+51\
   21c2f2+10090bdf2+7835cdf2+9997d2f2+12271bef2-4377cef2-4123def2-12796e2f2-\
   13503bf3+5456cf3-2360df3+5329ef3+9360f4+12880c3g-7763bcdg+15929c2dg-2586b\
   d2g+15985cd2g+10826d3g+10111bceg-14043c2eg-4415bdeg+691cdeg+15836d2eg+434\
   7be2g+6520ce2g+6764de2g+7956e3g+14069bcfg-2243c2fg-9042bdfg-4392cdfg+1098\
   2d2fg-15637befg-1789cefg+13273defg+4340e2fg-11245bf2g+2360cf2g+6883df2g-1\
   484ef2g+8898f3g-8467bcg2+2850c2g2-5378bdg2-15914cdg2+5796d2g2-9765beg2-15\
   123ceg2-9193deg2+5836e2g2+5788bfg2+14253cfg2-11274dfg2+15528efg2+5898f2g2\
   +7975bg3-4904cg3-912dg3+4581eg3+5109fg3-9975g4
↦ j[133]=bdeg8+93e3g8+3974bcfg8-15854c2fg8-13931bdfg8+3498cdfg8+4488d2fg8+8\
   170befg8-9194cefg8-15449defg8+3668e2fg8+7289bf2g8-4952cf2g8+11818df2g8-60\
   1ef2g8-6191f3g8-201bcg9+8719c2g9-1552bdg9+14722cdg9-11804d2g9-8274beg9-11\
   892ceg9+11030deg9-15263e2g9+13721bfg9-1426cfg9+1418dfg9+6885efg9-610f2g9-\
   3550bg10+3497cg10-4695dg10-7043eg10+7948fg10+1153g11-9163c4+11725bd3+1097\
   0cd3+9587d4+6315c3e+12556bcde-12145c2de+2046bd2e-12817cd2e-7328d3e+6805bc\
   e2+8037c2e2+13326bde2+8305cde2-7246d2e2-13433be3+4277ce3-10681de3+12336e4\
   +4305c3f+6002bcdf-326c2df+240bd2f+11134cd2f+8770d3f-3387bcef-3839c2ef+145\
   06bdef-2297cdef-3449d2ef-8229be2f+2632ce2f-10163de2f+15491e3f-5307bcf2+11\
   255c2f2+12907bdf2-6468cdf2+7367d2f2-2476bef2+12655cef2+15834def2+6195e2f2\
   +2209bf3+4695cf3-13533df3-7309ef3+11947f4-7436c3g+11577bcdg+8244c2dg-8776\
   bd2g+6596cd2g+8514d3g-7368bceg-6901c2eg-2452bdeg+2707cdeg+7898d2eg+8953be\
   2g+13292ce2g+12204de2g+11644e3g-10435bcfg+709c2fg+9812bdfg+1429cdfg-12911\
   d2fg-5704befg-3001cefg+5195defg+14948e2fg-11794bf2g-1605cf2g+6427df2g+969\
   2ef2g-5974f3g-5894bcg2+14401c2g2-5152bdg2-12877cdg2+6014d2g2+2739beg2+373\
   0ceg2-14553deg2-9185e2g2-5340bfg2-14551cfg2+14297dfg2+13377efg2-15560f2g2\
   +6700bg3-11940cg3-13622dg3-6003eg3-13345fg3+12846g4
↦ j[134]=c2eg8-7391e3g8-3890bcfg8+12893c2fg8+9119bdfg8-12175cdfg8-13105d2fg\
   8+7220befg8-10372cefg8+8626defg8+1956e2fg8-13291bf2g8+8131cf2g8+14558df2g\
   8+9848ef2g8+167f3g8+12224bcg9-12021c2g9-12100bdg9-2382cdg9-9921d2g9+12408\
   beg9-1999ceg9-15589deg9+14200e2g9+12311bfg9-3960cfg9-1929dfg9+7618efg9+57\
   4f2g9+1218bg10+14446cg10+9058dg10-8876eg10-9710fg10+11864g11+2290c4+8637b\
   d3-13186cd3-1340d4-11100c3e-6817bcde-14863c2de+412bd2e+7066cd2e+11871d3e-\
   15626bce2+7210c2e2+12892bde2-1393cde2+1839d2e2-7261be3-263ce3-4444de3+633\
   3e4-1251c3f+10764bcdf-9638c2df-14938bd2f-6531cd2f+15017d3f-10734bcef+2009\
   c2ef+12189bdef-6855cdef-4991d2ef-11132be2f-1976ce2f+12403de2f+2198e3f+113\
   87bcf2-13959c2f2+5537bdf2+339cdf2-6769d2f2-6259bef2-2086cef2+9459def2+120\
   79e2f2-3581bf3+9760cf3-734df3-9048ef3+7218f4+11731c3g+8907bcdg+2103c2dg+5\
   129bd2g+9645cd2g-1581d3g+3172bceg+15464c2eg+6304bdeg+9184cdeg-11977d2eg+1\
   2342be2g+13139ce2g-4626de2g+12761e3g-1423bcfg+6664c2fg-15639bdfg-1879cdfg\
   -10235d2fg-925befg-845cefg-5282defg-6407e2fg-9682bf2g+5336cf2g+8875df2g-7\
   011ef2g+5562f3g+11362bcg2+7186c2g2+3742bdg2-8076cdg2-437d2g2-5495beg2-116\
   49ceg2+3297deg2-6533e2g2+11919bfg2+2343cfg2+8732dfg2+12447efg2+12288f2g2-\
   8399bg3-7640cg3+12869dg3-13510eg3-8803fg3+9697g4
↦ j[135]=bceg8-7059e3g8+2345bcfg8+4854c2fg8+7647bdfg8-11395cdfg8+7056d2fg8-\
   5954befg8-9736cefg8+11306defg8-5241e2fg8+13788bf2g8+15054cf2g8+15796df2g8\
   +15495ef2g8+5083f3g8+7617bcg9-11636c2g9+11700bdg9-6590cdg9+1522d2g9+2405b\
   eg9+2854ceg9+13283deg9+12405e2g9-13817bfg9-11906cfg9+8740dfg9-12762efg9+6\
   459f2g9+2006bg10-4247cg10+12803dg10+10394eg10-10721fg10+12183g11-14102c4-\
   10473bd3-4155cd3-4363d4-7204c3e-9572bcde+7690c2de-1926bd2e-8797cd2e+7977d\
   3e-15985bce2-1354c2e2-758bde2+15535cde2+7952d2e2-1798be3+14510ce3-14242de\
   3-11448e4+10796c3f+6172bcdf-9102c2df-1703bd2f+3690cd2f-6302d3f+11850bcef-\
   382c2ef+4676bdef-864cdef+7128d2ef-4749be2f+15115ce2f+1913de2f+3946e3f-141\
   47bcf2-6764c2f2+15519bdf2-13125cdf2+305d2f2+3221bef2+15919cef2+8739def2-6\
   445e2f2+155bf3-9376cf3-3903df3+4265ef3+10837f4-12653c3g-2222bcdg-1109c2dg\
   -10493bd2g-763cd2g-5290d3g-9151bceg-2788c2eg-7860bdeg-14856cdeg-13492d2eg\
   +13200be2g+3579ce2g+13323de2g+13395e3g-1846bcfg+3371c2fg+5513bdfg+5008cdf\
   g-13108d2fg+4707befg-11366cefg-4149defg+3468e2fg-10543bf2g+4141cf2g+4524d\
   f2g-4710ef2g+7340f3g-1005bcg2+13052c2g2-5482bdg2+2476cdg2-13094d2g2-15983\
   beg2-3979ceg2+9189deg2+6452e2g2+1582bfg2-5642cfg2-3650dfg2+3208efg2+15914\
   f2g2-640bg3-4118cg3+7941dg3-5810eg3+13790fg3+8032g4
↦ j[136]=d3g8+6672e3g8+5297bcfg8+11913c2fg8-13099bdfg8+1197cdfg8+13057d2fg8\
   -3799befg8-3868cefg8-8287defg8+3554e2fg8-9726bf2g8-151cf2g8+3048df2g8-890\
   4ef2g8+2341f3g8+5073bcg9+1321c2g9-862bdg9-11514cdg9-15734d2g9+13893beg9+1\
   3497ceg9+15077deg9+12965e2g9-11273bfg9+13641cfg9+7421dfg9-13475efg9+2362f\
   2g9-9273bg10+6281cg10+8721dg10-12698eg10+12649fg10+8943g11-1588c4-15996bd\
   3+899cd3-11249d4+56c3e+9275bcde+3095c2de-2224bd2e-5597cd2e-7657d3e-12025b\
   ce2-11740c2e2-11307bde2-7421cde2+11047d2e2-2810be3-7200ce3-9241de3-12459e\
   4-14370c3f+2121bcdf+8732c2df+8878bd2f+13531cd2f+525d3f-11368bcef+13198c2e\
   f-2166bdef+7040cdef-9747d2ef-4596be2f-14881ce2f-9271de2f-5010e3f-8932bcf2\
   -12664c2f2+386bdf2-9953cdf2-6924d2f2-9426bef2+1079cef2+9795def2-13564e2f2\
   -10970bf3+14069cf3+8547df3-2312ef3+9449f4+4704c3g-720bcdg-4347c2dg-10117b\
   d2g+15893cd2g-8482d3g-14374bceg-12542c2eg+13125bdeg+1396cdeg-13590d2eg-35\
   1be2g+8038ce2g+4523de2g-2204e3g-5635bcfg-10654c2fg-1579bdfg-8460cdfg-1164\
   d2fg+4495befg-2993cefg+15698defg-12012e2fg-3143bf2g+9414cf2g+1123df2g-151\
   93ef2g-6831f3g+10629bcg2-13651c2g2+3339bdg2+9990cdg2-2913d2g2-77beg2-2878\
   ceg2-1687deg2-735e2g2-6323bfg2+8857cfg2+9721dfg2+11830efg2-110f2g2+9651bg\
   3+10090cg3-13262dg3+6429eg3+15865fg3+1893g4
↦ j[137]=cd2g8-11401e3g8-2467bcfg8+528c2fg8-9708bdfg8+840cdfg8-15037d2fg8+1\
   5924befg8+6718cefg8-11348defg8+10851e2fg8+11205bf2g8-6336cf2g8-3972df2g8+\
   5943ef2g8-9248f3g8-12238bcg9-9094c2g9-15989bdg9+15725cdg9+14925d2g9+3250b\
   eg9-9096ceg9+6977deg9-10001e2g9-4842bfg9+10348cfg9+6991dfg9+13409efg9-108\
   45f2g9+6727bg10+10051cg10+7031dg10+11353eg10-13954fg10+9938g11+2417c4+914\
   7bd3-204cd3+10392d4-368c3e+12279bcde+13684c2de-10703bd2e+9247cd2e+950d3e+\
   2596bce2+6593c2e2-3904bde2+8988cde2-5566d2e2+7154be3+3712ce3+8696de3+4407\
   e4-262c3f+10618bcdf+15952c2df+7780bd2f-6398cd2f-11822d3f-14641bcef-2195c2\
   ef+10853bdef-2270cdef+553d2ef-2729be2f-13810ce2f+14169de2f+8394e3f-5586bc\
   f2-8190c2f2-11606bdf2-3098cdf2+7284d2f2-5326bef2+13039cef2-9949def2+728e2\
   f2+147bf3-1268cf3+9240df3+7557ef3+13004f4+4987c3g-14065bcdg+10879c2dg-136\
   0bd2g-1415cd2g-7382d3g-2985bceg+14304c2eg+13453bdeg+6559cdeg+15419d2eg+33\
   90be2g-15088ce2g+13983de2g+13309e3g+11495bcfg+1314c2fg+8284bdfg-1495cdfg-\
   7091d2fg-1056befg+2451cefg-8715defg-15326e2fg-424bf2g+3511cf2g+7774df2g-4\
   060ef2g-4002f3g-1695bcg2+4633c2g2+1307bdg2+1376cdg2-15430d2g2+9702beg2-40\
   14ceg2+6444deg2+15916e2g2+13740bfg2+10009cfg2-4135dfg2-3535efg2-2894f2g2-\
   5724bg3+9997cg3-555dg3-8101eg3+11510fg3-6025g4
↦ j[138]=bd2g8+3699e3g8+9379bcfg8-9505c2fg8+8936bdfg8+13776cdfg8-8222d2fg8-\
   11579befg8+1517cefg8-3736defg8-936e2fg8-4940bf2g8+12144cf2g8-9064df2g8-72\
   4ef2g8-5698f3g8-1126bcg9+543c2g9+13679bdg9-2118cdg9+8073d2g9+12628beg9+79\
   69ceg9+4237deg9-4523e2g9+13495bfg9-6956cfg9+11435dfg9+11128efg9+10629f2g9\
   +3444bg10-1377cg10+10076dg10-15722eg10+1592fg10-2662g11-10623c4-8359bd3+9\
   86cd3-6555d4+12951c3e+13241bcde-8356c2de+117bd2e+6118cd2e-13485d3e+4919bc\
   e2+4891c2e2-9238bde2-1411cde2+10738d2e2+9052be3+2492ce3+11009de3-15695e4+\
   9923c3f+6925bcdf-7559c2df-12511bd2f+14320cd2f-9118d3f+15813bcef+14187c2ef\
   +15067bdef+1841cdef+6445d2ef-4076be2f-97ce2f-13821de2f+6769e3f+8178bcf2+7\
   03c2f2-3723bdf2-10746cdf2-11566d2f2+8887bef2-9435cef2-838def2+15687e2f2+1\
   4312bf3+4371cf3+15734df3-12611ef3+15855f4+57c3g+10790bcdg-13254c2dg-9392b\
   d2g-3818cd2g-3767d3g+3363bceg-5818c2eg-2013bdeg-3771cdeg+4701d2eg+15764be\
   2g+12884ce2g-9698de2g-9448e3g+6815bcfg-3964c2fg+15299bdfg-9384cdfg-14681d\
   2fg+2901befg-6876cefg-13838defg-5694e2fg+5793bf2g+1481cf2g+7777df2g-12941\
   ef2g-10397f3g-4701bcg2-10127c2g2+12370bdg2+1603cdg2+84d2g2-3062beg2-6215c\
   eg2+15071deg2-6802e2g2+4363bfg2+3722cfg2-15841dfg2+6789efg2-5164f2g2-796b\
   g3-11708cg3-15300dg3-7783eg3+4849fg3+2583g4
↦ j[139]=c2dg8+2397e3g8+8506bcfg8-13213c2fg8-2579bdfg8+10342cdfg8+5667d2fg8\
   +12507befg8-1816cefg8-6854defg8-14759e2fg8+7534bf2g8-5014cf2g8-10470df2g8\
   -6090ef2g8-5810f3g8+11084bcg9-10868c2g9-7761bdg9-3943cdg9-5435d2g9+2633be\
   g9-10892ceg9+4200deg9+5329e2g9+13359bfg9-4248cfg9+4195dfg9+1555efg9-3047f\
   2g9-5036bg10+8166cg10+175dg10-12123eg10-9702fg10-3352g11-10550c4+14612bd3\
   +14058cd3+1519d4-4896c3e-12170bcde-8397c2de-2798bd2e-9314cd2e+13639d3e-13\
   261bce2+13230c2e2+11670bde2+10794cde2+1512d2e2+9776be3-11351ce3-14297de3+\
   15175e4-15846c3f+7568bcdf-13081c2df+10121bd2f+6336cd2f-5519d3f-452bcef-12\
   90c2ef-9236bdef+4280cdef+4714d2ef+11162be2f-6767ce2f-10488de2f+8846e3f+10\
   37bcf2+2826c2f2-3474bdf2-11640cdf2+15131d2f2+7638bef2+4628cef2-1146def2-1\
   1391e2f2-2290bf3+4868cf3+682df3+2415ef3+15690f4-6759c3g+8188bcdg+15480c2d\
   g+6888bd2g+5192cd2g+8242d3g-5097bceg+6459c2eg-5869bdeg-4608cdeg+2947d2eg-\
   2006be2g-10655ce2g+4562de2g-9376e3g-11319bcfg-2595c2fg+656bdfg+787cdfg-46\
   02d2fg+6849befg-7299cefg-9757defg-3470e2fg+15265bf2g-14729cf2g-2386df2g-9\
   882ef2g-8625f3g-10500bcg2+19c2g2+7009bdg2+10325cdg2-103d2g2+1788beg2+9257\
   ceg2-13723deg2-3580e2g2+2526bfg2-11129cfg2-4621dfg2-8663efg2+5189f2g2+114\
   81bg3+6471cg3-13040dg3-7604eg3+8373fg3-4873g4
↦ j[140]=bcdg8+8735e3g8-156bcfg8+5596c2fg8+9211bdfg8-15345cdfg8-7304d2fg8-1\
   274befg8-13246cefg8-8459defg8+13967e2fg8-3376bf2g8+11846cf2g8+5360df2g8-9\
   159ef2g8-6995f3g8+12905bcg9+7204c2g9-5507bdg9-14702cdg9+3047d2g9-15beg9+1\
   1800ceg9+11105deg9-997e2g9-1667bfg9+9253cfg9+15542dfg9-8051efg9-14864f2g9\
   -1206bg10+11546cg10-1759dg10-2082eg10+6314fg10+4142g11-1028c4-9889bd3-812\
  6cd3+4588d4-11046c3e-2633bcde-8847c2de+2061bd2e+630cd2e+9325d3e+9358bce2-\
  13605c2e2-14731bde2+8516cde2-7621d2e2+4066be3+7646ce3+11595de3-13892e4+41\
  96c3f-15159bcdf+11469c2df+7314bd2f+13009cd2f+10679d3f+9618bcef-6684c2ef+7\
  749bdef+9805cdef-10681d2ef+11319be2f-13397ce2f-1843de2f-10477e3f-12675bcf\
  2+3806c2f2-8210bdf2-8315cdf2-421d2f2+13418bef2-9360cef2+14815def2+11323e2\
  f2-5507bf3-3560cf3+11176df3+13322ef3-719f4-13257c3g-11273bcdg+177c2dg-138\
  58bd2g+5106cd2g-10151d3g-391bceg+9807c2eg+1343bdeg+5355cdeg+6424d2eg-391b\
  e2g+13249ce2g-3979de2g-11553e3g-15917bcfg+6748c2fg-9524bdfg-6810cdfg+1108\
  d2fg-4333befg-4400cefg-537defg+11779e2fg-5224bf2g+7621cf2g-7833df2g+2914e\
  f2g+4339f3g-54bcg2-5906c2g2+15140bdg2-14683cdg2-10081d2g2+15708beg2-8869c\
  eg2+9867deg2+8483e2g2-7865bfg2-13986cfg2-15765dfg2+13184efg2-5330f2g2-379\
  bg3+10924cg3+3678dg3+10989eg3-10136fg3+4755g4
↦ j[141]=c3g8-8994e3g8+1963bcfg8-3802c2fg8+3723bdfg8-1632cdfg8+8315d2fg8-76\
  45befg8+9839cefg8+7903defg8+15271e2fg8-9561bf2g8-7546cf2g8-4530df2g8+2270\
  ef2g8-12928f3g8+2127bcg9-1230c2g9+7359bdg9+3603cdg9-155d2g9+2927beg9+2411\
  ceg9+3024deg9+10828e2g9+3230bfg9+13700cfg9-12547dfg9+766efg9+8569f2g9-355\
  3bg10+9500cg10+1290dg10-9104eg10+3318fg10-11149g11-2270c4-5835bd3-14186cd\
  3-15688d4-6602c3e-1601bcde-2593c2de+7916bd2e+774cd2e-11986d3e+7248bce2+25\
  84c2e2+3715bde2-10094cde2-9274d2e2-9059be3-12850ce3+6939de3-4735e4-9099c3\
  f+3439bcdf+9849c2df+11115bd2f+3518cd2f+2795d3f+3075bcef-15791c2ef-13561bd\
  ef+6950cdef-14354d2ef+5159be2f-9738ce2f+7314de2f+3428e3f-11573bcf2-7302c2\
  f2+5140bdf2-7344cdf2-13701d2f2-13991bef2-5779cef2+7657def2+5847e2f2-13761\
  bf3+7611cf3+8721df3+13374ef3-3941f4+3926c3g-13404bcdg-8499c2dg-14730bd2g-\
  3694cd2g+5686d3g-14617bceg+8120c2eg-8175bdeg-12114cdeg+12235d2eg+3275be2g\
  +3154ce2g+3558de2g-3396e3g+12743bcfg-11622c2fg+13853bdfg+5073cdfg-1936d2f\
  g+13275befg-10515cefg+3923defg+9246e2fg-6438bf2g-991cf2g+209df2g-15810ef2\
  g-15917f3g+376bcg2-13637c2g2+15981bdg2+8375cdg2+1230d2g2+10486beg2+2516ce\
  g2-4325deg2+12968e2g2-7424bfg2-1160cfg2+14346dfg2-2133efg2-918f2g2-15200b\
  g3-7231cg3-4656dg3+8706eg3-5589fg3-12162g4
↦ j[142]=f4g7+5347e3g8-6948bcfg8+15265c2fg8-9567bdfg8+3351cdfg8+7382d2fg8+4\
  749befg8+2805cefg8-12195defg8-2946e2fg8+5138bf2g8+6141cf2g8-12474df2g8+47\
  22ef2g8-12885f3g8+12588bcg9-6702c2g9+6797bdg9+7282cdg9-344d2g9+7011beg9+7\
  777ceg9-13029deg9-4540e2g9+4461bfg9-6498cfg9-1499dfg9-9817efg9+5964f2g9+4\
  761bg10+2745cg10+13213dg10+10006eg10-12864fg10+7631g11+9938c4-14149bd3+68\
  8cd3+7784d4+3215c3e-12818bcde+15898c2de+8280bd2e+15482cd2e+1551d3e-10205b\
  ce2+7190c2e2-14587bde2-9152cde2-15874d2e2+4076be3+3116ce3-4264de3+9167e4+\
  15033c3f-3765bcdf-11009c2df-15748bd2f+6242cd2f-1719d3f+15097bcef+1855c2ef\
  +1038bdef+4410cdef-13369d2ef-15560be2f+5169ce2f+11405de2f+5668e3f+9049bcf\
  2-15996c2f2+4691bdf2-500cdf2+14552d2f2+1495bef2-12089cef2-11814def2+8020e\
  2f2+13274bf3-7462cf3-3159df3-14366ef3+3002f4-11979c3g-4427bcdg+5773c2dg-1\
  5242bd2g+10030cd2g-10231d3g-5363bceg+2274c2eg-10707bdeg+499cdeg-200d2eg+1\
  074be2g+5687ce2g+11636de2g+7534e3g+4617bcfg-10217c2fg-13987bdfg-9426cdfg-\
  13903d2fg-7294befg+5962cefg+10293defg-6540e2fg+10983bf2g-1579cf2g+3786df2\
  g-746ef2g+3681f3g-12894bcg2-1066c2g2+15245bdg2-8117cdg2+7468d2g2-15703beg\
  2+1447ceg2-705deg2-5279e2g2-3134bfg2+2534cfg2+6596dfg2-2891efg2+4522f2g2-\
  6332bg3-4829cg3+14362dg3+8509eg3-2583fg3+9253g4
↦ j[143]=ef3g7+7882e3g8+5264bcfg8+11779c2fg8+12561bdfg8-2932cdfg8+8081d2fg8\
  -4693befg8-11098cefg8-4234defg8+5833e2fg8-2376bf2g8+363cf2g8-3648df2g8-52\
  20ef2g8-7486f3g8+7973bcg9-14396c2g9+2448bdg9+9959cdg9-8720d2g9-11133beg9+\
  15411ceg9+4635deg9+2407e2g9-965bfg9-9160cfg9-2057dfg9-14201efg9-845f2g9-7\
  513bg10+10909cg10+1834dg10+3947eg10-10111fg10+4769g11+7562c4-14873bd3+502\
  9cd3+15585d4+10380c3e+13012bcde+9586c2de-10814bd2e+8712cd2e-8322d3e+1415b\
  ce2-790c2e2-12738bde2+2860cde2+14325d2e2+5463be3+2348ce3-788de3-6617e4+41\
   76c3f+15328bcdf+10316c2df+11180bd2f-446cd2f-7859d3f-12081bcef-390c2ef+578\
   1bdef+10288cdef-15887d2ef+15914be2f+14864ce2f-2320de2f-11220e3f+4769bcf2-\
   14802c2f2-5168bdf2-14038cdf2-723d2f2+1228bef2-770cef2-13593def2-4227e2f2-\
   4756bf3-910cf3-14802df3+1903ef3+14076f4-9611c3g-14612bcdg+4600c2dg-4656bd\
   2g-11146cd2g-1103d3g+13813bceg+13611c2eg-6595bdeg+2874cdeg+8580d2eg-671be\
   2g-13790ce2g-7530de2g+9208e3g-9292bcfg+13449c2fg-14277bdfg+13568cdfg+1384\
   7d2fg-5572befg-14126cefg-5166defg-12028e2fg+11697bf2g-2362cf2g-7153df2g-1\
   2274ef2g-10738f3g+5127bcg2-5695c2g2-9577bdg2+612cdg2+1326d2g2+568beg2+153\
   12ceg2-431deg2-1699e2g2-1743bfg2+11036cfg2-10743dfg2-4298efg2+6544f2g2-11\
   439bg3+13132cg3-4002dg3-8296eg3-6650fg3+13934g4
↦ j[144]=df3g7-4089e3g8-2423bcfg8-3095c2fg8+7750bdfg8+657cdfg8-2875d2fg8-12\
   085befg8-2949cefg8-2030defg8+5488e2fg8-15480bf2g8-15278cf2g8-6592df2g8-90\
   24ef2g8+15483f3g8-2780bcg9+2913c2g9-7323bdg9-9022cdg9+6672d2g9+2766beg9-3\
   825ceg9-11567deg9-6993e2g9+2236bfg9+98cfg9+13877dfg9+4035efg9-1214f2g9-11\
   32bg10+15539cg10+12829dg10+5611eg10+10688fg10-7752g11+11975c4-7188bd3-411\
   4cd3-5016d4+3391c3e+2721bcde+13406c2de+11441bd2e-15039cd2e-13694d3e+3512b\
   ce2-4844c2e2-2782bde2+3946cde2+13907d2e2+3548be3-4726ce3+5515de3+469e4-18\
   55c3f-11404bcdf+12633c2df-14083bd2f+14187cd2f+1544d3f-7512bcef+15747c2ef-\
   13814bdef+2412cdef+11616d2ef+7232be2f-1084ce2f-7462de2f-14292e3f+2778bcf2\
   +14973c2f2+4570bdf2+12539cdf2-12110d2f2-431bef2+10463cef2+4933def2-7120e2\
   f2-10037bf3-2000cf3+6408df3+12396ef3-4160f4-3943c3g+2359bcdg+9658c2dg-614\
   0bd2g-14971cd2g-14939d3g+3255bceg+6530c2eg+10958bdeg+4907cdeg-9799d2eg-33\
   9be2g+544ce2g-14559de2g-7623e3g+5775bcfg+4857c2fg-8193bdfg+7043cdfg+6328d\
   2fg-10842befg+9562cefg-3640defg+9146e2fg+9920bf2g+10802cf2g-11163df2g+284\
   1ef2g+1549f3g+5228bcg2-367c2g2+13882bdg2-15496cdg2-12879d2g2+11856beg2-91\
   92ceg2-1364deg2+12164e2g2-3432bfg2-14719cfg2-2696dfg2-14999efg2-5290f2g2-\
   6370bg3-9016cg3-672dg3-630eg3-8637fg3+8133g4
↦ j[145]=cf3g7-3194e3g8-734bcfg8+11141c2fg8+14845bdfg8-11888cdfg8+14673d2fg\
   8+6250befg8-15736cefg8-2165defg8+12767e2fg8+9713bf2g8+7888cf2g8+4960df2g8\
   +15984ef2g8+8776f3g8-15448bcg9-9948c2g9+12885bdg9+8065cdg9-1728d2g9-13398\
   beg9-15520ceg9+12173deg9+3715e2g9-15366bfg9-4161cfg9-15721dfg9-1676efg9-1\
   2526f2g9-11435bg10-10251cg10+681dg10+1481eg10+8303fg10-3605g11+13212c4-92\
   97bd3-13518cd3-9927d4-14510c3e+3232bcde+4727c2de+425bd2e+10527cd2e-48d3e-\
   14537bce2-10685c2e2+10991bde2+4279cde2-11836d2e2-12002be3+12926ce3+9598de\
   3+1353e4-3282c3f+15954bcdf+6134c2df-6283bd2f+1440cd2f+7483d3f-5506bcef-11\
   558c2ef-1044bdef-2253cdef-5052d2ef+2332be2f+4821ce2f-8850de2f-11976e3f+87\
   86bcf2-13131c2f2-3967bdf2-1792cdf2-11973d2f2-5551bef2-15394cef2+11449def2\
   -14569e2f2-686bf3+11283cf3+119df3-6855ef3+6385f4+10639c3g-5280bcdg-10816c\
   2dg-6231bd2g-1855cd2g-1375d3g+15115bceg-4577c2eg-8089bdeg+12352cdeg+7382d\
   2eg-1866be2g-8415ce2g+5747de2g-4604e3g-3793bcfg+2998c2fg+11400bdfg-761cdf\
   g+6671d2fg+715befg-13480cefg-33defg+8856e2fg+7421bf2g-12529cf2g-10896df2g\
   +5788ef2g-7543f3g+13397bcg2+10073c2g2-372bdg2+1131cdg2-2737d2g2-6237beg2+\
   7319ceg2+2718deg2+15159e2g2+3344bfg2+12949cfg2+6942dfg2+12931efg2-9833f2g\
   2+10317bg3-15922cg3+14325dg3-9799eg3-10677fg3+8853g4
↦ j[146]=bf3g7+5959e3g8+13273bcfg8+2481c2fg8-532bdfg8-734cdfg8+147d2fg8+516\
   0befg8+15411cefg8+4902defg8-4464e2fg8-7533bf2g8+7734cf2g8+14459df2g8-5240\
   ef2g8+6337f3g8+3752bcg9-10262c2g9-592bdg9+10647cdg9-12638d2g9+12982beg9-9\
   958ceg9+3742deg9-10923e2g9-233bfg9+14086cfg9-6669dfg9-3513efg9-7048f2g9-1\
   4014bg10+53cg10+4049dg10+2755eg10-12428fg10+1492g11+11278c4-11395bd3-1196\
   1cd3-14817d4-9508c3e-10045bcde-12406c2de-12009bd2e+7207cd2e+15469d3e+1103\
   7bce2+9557c2e2-11388bde2-13384cde2-7270d2e2+12805be3-3254ce3-2667de3+1599\
   3e4-15178c3f+3062bcdf-5789c2df-10183bd2f+11510cd2f-15791d3f-526bcef-4198c\
   2ef+3835bdef-2312cdef+5473d2ef+6925be2f+8281ce2f+2689de2f-9499e3f+2700bcf\
   2-869c2f2-4184bdf2+5561cdf2+2974d2f2+1769bef2-8422cef2-12027def2-6558e2f2\
   -13765bf3+10708cf3-286df3+4856ef3+13141f4-9660c3g-11124bcdg-7410c2dg+8022\
   bd2g-2780cd2g+10688d3g-10563bceg-13466c2eg-7377bdeg-5844cdeg+6043d2eg-142\
   19be2g-10653ce2g-4251de2g+11584e3g-15165bcfg+2521c2fg+8765bdfg+120cdfg-90\
   62d2fg+9479befg-3278cefg+7623defg-5401e2fg-13454bf2g-10814cf2g-15048df2g+\
   4521ef2g+8373f3g-5675bcg2+9656c2g2-8592bdg2+99cdg2+11967d2g2+10060beg2-84\
   06ceg2-8976deg2+11079e2g2+4576bfg2-6882cfg2-5447dfg2-13801efg2-3919f2g2+3\
   59bg3+12702cg3-13181dg3+8583eg3+13333fg3-3497g4
↦ j[147]=e2f2g7-444e3g8-8722bcfg8+2015c2fg8+940bdfg8+14498cdfg8+15383d2fg8+\
   8395befg8+7901cefg8+12999defg8+13589e2fg8+13885bf2g8-8486cf2g8-14375df2g8\
   +11829ef2g8+3778f3g8-10539bcg9+4195c2g9+14940bdg9-14675cdg9+13965d2g9-136\
   92beg9+7302ceg9-2262deg9-6222e2g9-337bfg9+4404cfg9+4580dfg9+6157efg9+9593\
   f2g9+4353bg10-2194cg10-13723dg10-15940eg10-7417fg10+10375g11-11785c4-6150\
   bd3-14283cd3+9400d4-6692c3e-13650bcde+5358c2de+6579bd2e+11837cd2e-13718d3\
   e-14402bce2+5655c2e2-15838bde2-9151cde2-7788d2e2-6413be3+4733ce3-4348de3+\
   4448e4-1070c3f-11252bcdf+2209c2df+15617bd2f-3381cd2f+13844d3f+9861bcef+14\
   83c2ef-4707bdef+7143cdef+7372d2ef-3357be2f-6662ce2f-3225de2f-1323e3f+398b\
   cf2-9863c2f2+14181bdf2+9994cdf2+5286d2f2-8753bef2+12382cef2-2593def2+1220\
   5e2f2+3196bf3-463cf3+8548df3+12665ef3-5059f4-7195c3g-8131bcdg-3837c2dg-57\
   79bd2g-1355cd2g+9550d3g-15557bceg+1194c2eg+2713bdeg+12941cdeg-8988d2eg-75\
   74be2g-4948ce2g+15686de2g-3764e3g-2430bcfg-3103c2fg-15404bdfg+14649cdfg+2\
   756d2fg+9053befg+10053cefg-1845defg+15938e2fg-13703bf2g-6635cf2g+15623df2\
   g-10146ef2g-13665f3g+11673bcg2+11806c2g2-5789bdg2-11765cdg2+15522d2g2+145\
   96beg2+11513ceg2-8606deg2+359e2g2+529bfg2+1998cfg2+2771dfg2-903efg2+966f2\
   g2-2736bg3+738cg3+8063dg3-12939eg3-8475fg3-5929g4
↦ j[148]=def2g7+11262e3g8-14352bcfg8+14534c2fg8-15135bdfg8-9378cdfg8+10994d\
   2fg8+8059befg8+5234cefg8-10540defg8-9217e2fg8+8079bf2g8+12659cf2g8+14116d\
   f2g8+15765ef2g8-14277f3g8+4441bcg9+279c2g9-2399bdg9+15912cdg9+8104d2g9-15\
   108beg9+6856ceg9+11073deg9+4963e2g9-1448bfg9+10369cfg9+11516dfg9+1097efg9\
   -766f2g9-8530bg10+9538cg10+15559dg10+134eg10-7685fg10-15502g11+13785c4-82\
   62bd3-9141cd3-15010d4+3047c3e+11926bcde+12550c2de+12766bd2e-12066cd2e-132\
   92d3e+468bce2+5936c2e2+7168bde2+9375cde2+4363d2e2-12498be3-9968ce3-361de3\
   -4042e4-8724c3f-8861bcdf-15779c2df+11560bd2f+12777cd2f-3875d3f-12627bcef-\
   3947c2ef+3842bdef+10537cdef+12031d2ef-56be2f+2425ce2f+10007de2f-3243e3f+4\
   657bcf2-11530c2f2-11619bdf2+46cdf2-707d2f2+9269bef2+247cef2+6408def2+1453\
   7e2f2+12143bf3-4787cf3-11852df3+2146ef3-12636f4+1948c3g-10342bcdg-13925c2\
   dg-4867bd2g-15454cd2g+4200d3g-7729bceg+11003c2eg-2901bdeg+6556cdeg-7072d2\
   eg-9956be2g-3419ce2g-6647de2g+15098e3g-2789bcfg-8269c2fg-8730bdfg+13600cd\
   fg+13070d2fg+6021befg-11722cefg-14327defg+5921e2fg+2205bf2g-11314cf2g+932\
   4df2g+8285ef2g-6113f3g-14543bcg2-11189c2g2-7101bdg2+10170cdg2+5331d2g2-30\
   87beg2-9023ceg2-14500deg2-11685e2g2-14401bfg2-12221cfg2-5018dfg2-11511efg\
   2+15047f2g2-1272bg3+4879cg3-2120dg3+7295eg3+2636fg3+13315g4
↦ j[149]=cef2g7+13071e3g8+7150bcfg8+9613c2fg8-12509bdfg8+4927cdfg8+15390d2f\
   g8+3344befg8+655cefg8+3650defg8-15236e2fg8-12746bf2g8+13858cf2g8+7351df2g\
   8+10299ef2g8+14600f3g8-7450bcg9-3007c2g9-7694bdg9-15822cdg9+3779d2g9+3495\
   beg9+1044ceg9+208deg9-9539e2g9+14319bfg9-12848cfg9+14305dfg9+7169efg9+205\
   7f2g9-8915bg10+5895cg10-6235dg10-206eg10+9369fg10-14005g11-6528c4+15393bd\
   3+8782cd3+15500d4+3932c3e-15828bcde-5402c2de+5106bd2e-8723cd2e+14799d3e+1\
   1134bce2-10085c2e2-4207bde2-15573cde2-2697d2e2+2561be3+15531ce3-1393de3-7\
   936e4+10640c3f+4504bcdf-11885c2df+5176bd2f+15354cd2f-15417d3f-13866bcef+4\
   37c2ef+13200bdef-2369cdef+5879d2ef-2131be2f+8847ce2f+8889de2f+4623e3f+116\
   37bcf2+4617c2f2+6168bdf2-327cdf2+3661d2f2+8484bef2-11142cef2-661def2-6620\
   e2f2-7860bf3+9414cf3+9646df3-10113ef3-3136f4+12238c3g+2676bcdg+15992c2dg-\
    1009bd2g+2786cd2g+35d3g+13434bceg-13737c2eg+3634bdeg-1788cdeg-9965d2eg+12\
    671be2g+10167ce2g-12597de2g-882e3g-11899bcfg+14602c2fg-14881bdfg+13816cdf\
    g-2614d2fg-10700befg+8989cefg+1248defg+12319e2fg-1420bf2g+15532cf2g-14146\
    df2g-10979ef2g+12328f3g-7593bcg2+6533c2g2-93bdg2-3099cdg2+7671d2g2+11251b\
    eg2+10067ceg2-15987deg2-2654e2g2+6865bfg2-4065cfg2+429dfg2+6631efg2-12776\
    f2g2+10602bg3-11446cg3+5874dg3+3263eg3-2310fg3+5650g4
↦ j[150]=bef2g7-827e3g8+4931bcfg8-5331c2fg8-9740bdfg8-14125cdfg8+1365d2fg8+\
    31befg8-10041cefg8-11843defg8-11535e2fg8-10985bf2g8-11389cf2g8+15542df2g8\
    -14207ef2g8-3595f3g8-11469bcg9+5104c2g9-13859bdg9-15635cdg9-3547d2g9+1336\
    1beg9-6227ceg9+7411deg9-3159e2g9-5958bfg9-3192cfg9+3271dfg9+12571efg9-457\
    4f2g9-8187bg10-15352cg10+1335dg10+3490eg10-11596fg10+3116g11+3268c4+12157\
    bd3-4419cd3-9628d4-434c3e+14231bcde-6987c2de-3440bd2e+9917cd2e-2897d3e+10\
    429bce2+704c2e2+3888bde2-10258cde2+12274d2e2-8774be3-13027ce3-15097de3-13\
    519e4-4957c3f-3199bcdf+292c2df-11043bd2f+5668cd2f+1386d3f-2842bcef+10302c\
    2ef-12814bdef+12826cdef-1707d2ef+12206be2f-11733ce2f-2054de2f+3936e3f-930\
    0bcf2+4582c2f2-10686bdf2-12300cdf2+1719d2f2-7525bef2-5325cef2+555def2-134\
    11e2f2+6814bf3+11093cf3+3582df3-1349ef3+4021f4-13659c3g+943bcdg+9080c2dg+\
    11944bd2g-6957cd2g+1567d3g+6847bceg-4505c2eg-7272bdeg+8582cdeg+5808d2eg+1\
    0420be2g+11486ce2g+13105de2g+5335e3g-15614bcfg-15835c2fg+4431bdfg+8866cdf\
    g-5299d2fg-12653befg+9912cefg+3945defg-157e2fg-10516bf2g+8612cf2g-1668df2\
    g-1235ef2g-7260f3g+13480bcg2-2517c2g2-14538bdg2-14366cdg2+5151d2g2+6689be\
    g2+8127ceg2+11705deg2-13518e2g2-980bfg2-13730cfg2+444dfg2-5281efg2-2222f2\
    g2-5415bg3+7934cg3+12086dg3-11712eg3-13648fg3+9759g4
↦ j[151]=d2f2g7-12384e3g8-6496bcfg8-1407c2fg8+1963bdfg8-3399cdfg8-9936d2fg8\
    +10817befg8-3940cefg8+4620defg8+3781e2fg8+13938bf2g8-4049cf2g8+9970df2g8-\
    14513ef2g8-7448f3g8-1591bcg9-7008c2g9+4030bdg9+15765cdg9+8897d2g9+9699beg\
    9+1061ceg9-3345deg9+9852e2g9+14738bfg9+5929cfg9-13782dfg9+3435efg9+1985f2\
    g9-3851bg10+12245cg10+8753dg10+15327eg10-3169fg10-8046g11+13941c4+11194bd\
    3-4632cd3+317d4-14153c3e-12103bcde-5933c2de+7979bd2e+5336cd2e+9362d3e-134\
    62bce2-3282c2e2+3712bde2-3395cde2-5528d2e2+5738be3-4556ce3+10520de3+2373e\
    4+9565c3f-14453bcdf+13101c2df+8718bd2f+15250cd2f-8628d3f+212bcef+12104c2e\
    f-5346bdef-1991cdef+5724d2ef-11000be2f-7155ce2f+15931de2f+1972e3f+4514bcf\
    2-191c2f2-2930bdf2+13147cdf2+13525d2f2-5266bef2-5946cef2+5097def2+10002e2\
    f2-12229bf3+15020cf3+464df3-2334ef3+7177f4-226c3g-1482bcdg+11224c2dg+4740\
    bd2g-14724cd2g-2902d3g+11239bceg-13707c2eg+11135bdeg+999cdeg+6941d2eg+374\
    2be2g-187ce2g-1554de2g-151e3g-12557bcfg+5970c2fg-6070bdfg-3482cdfg-8065d2\
    fg+9627befg-14030cefg-8573defg+13133e2fg-1109bf2g+13503cf2g-10224df2g-715\
    8ef2g+4673f3g+5260bcg2+2167c2g2-11068bdg2-8011cdg2-2810d2g2+8043beg2+9368\
    ceg2+13961deg2-7125e2g2-13750bfg2-5148cfg2+2166dfg2-10079efg2+11194f2g2+1\
    5728bg3+9714cg3+7833dg3-1375eg3+13079fg3+219g4
↦ j[152]=cdf2g7+11277e3g8-3290bcfg8-2199c2fg8-1274bdfg8+10580cdfg8+11208d2f\
    g8+9211befg8-12547cefg8-12321defg8-6658e2fg8-4164bf2g8-10711cf2g8+578df2g\
    8-2814ef2g8+13812f3g8-11118bcg9+1890c2g9-2284bdg9-7987cdg9-13777d2g9-3359\
    beg9-920ceg9+2395deg9-5980e2g9-13660bfg9+2136cfg9-10340dfg9-3231efg9-829f\
    2g9+15975bg10+9744cg10-15005dg10+3872eg10+3599fg10-14487g11-891c4-10115bd\
    3-12502cd3-4633d4-7239c3e+2358bcde+2524c2de-7512bd2e+6329cd2e-12523d3e-69\
    46bce2-14199c2e2-14783bde2-4961cde2-1836d2e2+7247be3-1710ce3+11781de3+979\
    6e4+2487c3f-9022bcdf+6296c2df-11579bd2f-14839cd2f-6435d3f+9286bcef+9025c2\
    ef+2341bdef+2728cdef-2559d2ef-5610be2f+14366ce2f-8959de2f+10915e3f+9426bc\
    f2-15351c2f2+7505bdf2-457cdf2-5890d2f2-6373bef2-4367cef2-11622def2-15084e\
    2f2+9137bf3-4704cf3-13554df3-3847ef3+4027f4+4830c3g+5475bcdg-7949c2dg-144\
    55bd2g-14795cd2g-5816d3g+1680bceg-7847c2eg+5373bdeg-5002cdeg-14476d2eg+92\
    38be2g-13317ce2g+2168de2g+2037e3g+10841bcfg-8486c2fg+13193bdfg+879cdfg+11\
   586d2fg+10786befg-13631cefg-12515defg+4624e2fg-14892bf2g+2191cf2g+10114df\
   2g-14787ef2g-6402f3g+6377bcg2+5489c2g2+4683bdg2-13293cdg2+4972d2g2-3559be\
   g2-13280ceg2+11722deg2+1593e2g2-11696bfg2-7617cfg2-14703dfg2-11725efg2+21\
   96f2g2+15559bg3-5738cg3-11028dg3+15983eg3+15246fg3-12645g4
↦ j[153]=bdf2g7-8697e3g8+14249bcfg8-13545c2fg8-5477bdfg8-12802cdfg8-4883d2f\
   g8+2493befg8+14602cefg8-14977defg8-8894e2fg8-2620bf2g8-4529cf2g8+3681df2g\
   8-2280ef2g8+1303f3g8+618bcg9+7936c2g9-14811bdg9+9915cdg9+12370d2g9+10430b\
   eg9+7754ceg9+1490deg9+4546e2g9+3418bfg9+11078cfg9-13047dfg9+8584efg9-1228\
   9f2g9+12293bg10-12850cg10+10587dg10-5792eg10-4155fg10+9753g11+2900c4-6893\
   bd3-2988cd3-3716d4-14816c3e-5209bcde-13772c2de+10659bd2e-8860cd2e+8013d3e\
   +10423bce2+14954c2e2+6828bde2+15578cde2+13574d2e2+8023be3+6279ce3+13529de\
   3+11233e4+1661c3f-6584bcdf-10447c2df-7070bd2f+11002cd2f-6992d3f+3176bcef-\
   3706c2ef-13628bdef-10293cdef-1708d2ef+12645be2f-1314ce2f-1662de2f+3928e3f\
   -7116bcf2+11676c2f2+1044bdf2+468cdf2-14253d2f2-3992bef2-8841cef2+14666def\
   2+5741e2f2+14506bf3-13778cf3-5641df3-6432ef3-6453f4+2029c3g+15614bcdg-151\
   97c2dg-4745bd2g+14174cd2g+5773d3g-908bceg-123c2eg+13724bdeg+6573cdeg-6878\
   d2eg-15573be2g-6291ce2g-1678de2g-15176e3g-3158bcfg+13026c2fg-8868bdfg+116\
   83cdfg+8490d2fg+9742befg+15945cefg-13149defg+4050e2fg-2581bf2g-1536cf2g+3\
   493df2g-12476ef2g+10061f3g+7287bcg2-908c2g2+11321bdg2+6656cdg2-3056d2g2-6\
   803beg2+4770ceg2-15574deg2+6257e2g2+1306bfg2+4068cfg2-10113dfg2+13493efg2\
   -7775f2g2+10242bg3-14906cg3-14819dg3-5844eg3+1687fg3-3121g4
↦ j[154]=c2f2g7-15082e3g8-10670bcfg8+737c2fg8+623bdfg8+12040cdfg8+7309d2fg8\
   +2346befg8+8339cefg8+2148defg8+1350e2fg8+10702bf2g8+12318cf2g8+2837df2g8-\
   2302ef2g8+7370f3g8-6493bcg9+4751c2g9-3817bdg9+262cdg9+15641d2g9+4602beg9+\
   11174ceg9+691deg9-12542e2g9+9376bfg9+10053cfg9+4203dfg9+1038efg9+11247f2g\
   9-15631bg10+12650cg10+9989dg10+13464eg10+5916fg10-8004g11-6278c4+15865bd3\
   +3458cd3-5555d4+12811c3e+3922bcde+11003c2de-13370bd2e+6388cd2e-854d3e+125\
   1bce2-4679c2e2-10025bde2+256cde2-13447d2e2+12432be3+6267ce3-3463de3-3142e\
   4-187c3f-15550bcdf-2775c2df-2643bd2f+5270cd2f-2704d3f+14880bcef+5188c2ef-\
   11149bdef+9158cdef+1150d2ef-6688be2f-3500ce2f+7665de2f+6717e3f+9357bcf2+1\
   5389c2f2-14564bdf2-15723cdf2+8410d2f2+7145bef2-2258cef2-1768def2+9391e2f2\
   +4610bf3+5813cf3+8926df3-14024ef3+12f4-10404c3g-13592bcdg-6303c2dg-7848bd\
   2g-9548cd2g+10922d3g-3111bceg+4785c2eg-770bdeg-10767cdeg-1264d2eg+10675be\
   2g-4085ce2g-14021de2g-12428e3g-12745bcfg-4464c2fg+2474bdfg+8060cdfg+15449\
   d2fg-15532befg-791cefg+5289defg+7567e2fg-7601bf2g+9144cf2g+5137df2g-2534e\
   f2g+7104f3g-4892bcg2+9158c2g2+14060bdg2+1685cdg2+12490d2g2-10665beg2+2336\
   ceg2+7443deg2-2118e2g2+6678bfg2-1373cfg2+9611dfg2-11023efg2-14740f2g2-764\
   8bg3-10218cg3-10945dg3+11883eg3-14915fg3+13380g4
↦ j[155]=bcf2g7+15111e3g8-2298bcfg8-7801c2fg8+766bdfg8-5034cdfg8-9855d2fg8-\
   7096befg8+14554cefg8-3198defg8+14449e2fg8-8556bf2g8+201cf2g8-8877df2g8+15\
   824ef2g8-431f3g8+9025bcg9-5319c2g9+4221bdg9+2294cdg9+12478d2g9+6868beg9-6\
   231ceg9-2923deg9-4365e2g9-7157bfg9+2410cfg9+8920dfg9-6511efg9+8733f2g9-12\
   480bg10-7342cg10-1939dg10+9894eg10-5057fg10+8381g11+6341c4+5279bd3+13168c\
   d3+14610d4+13945c3e+3589bcde-10646c2de+11277bd2e-6842cd2e+1405d3e-11089bc\
   e2+12500c2e2-9612bde2+5316cde2-11029d2e2-2285be3-15952ce3-3532de3-11220e4\
   -2157c3f+2847bcdf-1918c2df-4485bd2f+3555cd2f+1138d3f+12296bcef-15271c2ef-\
   7905bdef-8990cdef-9689d2ef-13907be2f-236ce2f-6395de2f+6545e3f-12045bcf2+1\
   4209c2f2+14145bdf2-1832cdf2-12863d2f2+3640bef2-167cef2-5719def2+9451e2f2+\
   7187bf3+9401cf3+10816df3+6766ef3+13033f4-5177c3g+636bcdg-10929c2dg-5948bd\
   2g+8004cd2g+7002d3g+2401bceg-7259c2eg-874bdeg-5567cdeg+202d2eg+4154be2g+1\
   394ce2g-10005de2g+8658e3g-10058bcfg+11712c2fg-12387bdfg+6373cdfg-12039d2f\
   g+2939befg+5204cefg+14287defg+15402e2fg+8206bf2g+2249cf2g+6142df2g+7631ef\
   2g+4719f3g+13386bcg2-6957c2g2+8343bdg2+3379cdg2-10445d2g2-5330beg2+7442ce\
   g2+4493deg2+115e2g2+6944bfg2-10749cfg2-15538dfg2-13314efg2+15549f2g2+5362\
   bg3-8576cg3-13659dg3+14848eg3-2554fg3-4138g4
↦ j[156]=e3fg7+15289e3g8+6150bcfg8+6100c2fg8+6231bdfg8+11620cdfg8+12721d2fg\
   8+6733befg8-12719cefg8-4896defg8-1338e2fg8+11461bf2g8-3916cf2g8+6347df2g8\
   +6025ef2g8-6637f3g8+2297bcg9-6180c2g9+10445bdg9+9658cdg9+1606d2g9-3130beg\
   9-8887ceg9+3008deg9+15523e2g9+12242bfg9+12591cfg9+4017dfg9+8740efg9+6634f\
   2g9-746bg10+3572cg10-4148dg10+15412eg10-3627fg10-6871g11-15607c4-7700bd3-\
   14094cd3+2468d4+5707c3e-5930bcde-11070c2de-12010bd2e-13444cd2e-2623d3e+53\
   47bce2+12985c2e2-9693bde2+2090cde2-4339d2e2-14033be3+13955ce3-14695de3-91\
   94e4-7116c3f-10911bcdf-14538c2df+6584bd2f-9579cd2f+15480d3f-2213bcef-7425\
   c2ef+6340bdef-9729cdef+14132d2ef-1398be2f+14798ce2f+7717de2f-5723e3f+8410\
   bcf2-11060c2f2-2126bdf2+719cdf2-10631d2f2+7519bef2-12222cef2+14280def2-14\
   233e2f2+5090bf3+6078cf3+13578df3-7743ef3-13681f4-10496c3g+8720bcdg+15671c\
   2dg+7506bd2g-10257cd2g+11303d3g-12075bceg+13868c2eg+13622bdeg+3976cdeg+24\
   16d2eg-9876be2g-13080ce2g-15465de2g+5096e3g+5531bcfg-8218c2fg+13241bdfg-2\
   799cdfg-15193d2fg-6467befg+7053cefg-7122defg-789e2fg-9829bf2g+11336cf2g-2\
   363df2g-11739ef2g+11099f3g-8576bcg2+15113c2g2+4523bdg2+5079cdg2+13833d2g2\
   -14631beg2-7799ceg2-1281deg2+11013e2g2-8276bfg2-11690cfg2+2768dfg2-406efg\
   2-13769f2g2-8372bg3-11780cg3-6342dg3+382eg3+543fg3-6850g4
↦ j[157]=de2fg7-2122e3g8-6166bcfg8-3112c2fg8+852bdfg8+12428cdfg8-14443d2fg8\
   -6586befg8-8818cefg8+5266defg8+4559e2fg8-14265bf2g8-15294cf2g8+1844df2g8+\
   5814ef2g8-10623f3g8+15234bcg9+806c2g9+2938bdg9-2672cdg9+7737d2g9+10799beg\
   9-1068ceg9-3207deg9+15933e2g9+1364bfg9+9071cfg9+15515dfg9-6350efg9+6467f2\
   g9+3374bg10+2589cg10+6815dg10-5694eg10-1575fg10-12163g11+12586c4-3378bd3-\
   2668cd3+12436d4+6290c3e-2072bcde-6798c2de-5885bd2e+5664cd2e+3476d3e-5124b\
   ce2-2872c2e2-6918bde2+5596cde2+12720d2e2-448be3-3646ce3+6160de3+14957e4-3\
   087c3f-264bcdf-3089c2df-10135bd2f+13030cd2f-9381d3f+7143bcef+4159c2ef+841\
   3bdef+4558cdef+754d2ef-625be2f+4310ce2f-8191de2f+15488e3f-13249bcf2+14053\
   c2f2+4708bdf2+1641cdf2+5913d2f2+8598bef2-386cef2+10223def2+5013e2f2+13558\
   bf3-14305cf3+754df3-9457ef3+11548f4-10034c3g-11745bcdg-12645c2dg+4892bd2g\
   +15610cd2g-13939d3g+15173bceg-14010c2eg+11061bdeg-4945cdeg+12343d2eg+1270\
   3be2g+1985ce2g+7687de2g-1664e3g-8589bcfg-579c2fg-11094bdfg+14203cdfg+1514\
   0d2fg-9190befg+392cefg+6651defg+2979e2fg-7518bf2g-4115cf2g-1451df2g-5876e\
   f2g+14268f3g-14736bcg2+8675c2g2+9773bdg2-7622cdg2+1839d2g2-3284beg2+9853c\
   eg2-2327deg2-3286e2g2+9612bfg2+13089cfg2-12676dfg2-1694efg2+8424f2g2+1076\
   bg3+6048cg3+9294dg3+8373eg3-7646fg3-7482g4
↦ j[158]=ce2fg7-8608e3g8+12774bcfg8-5462c2fg8+9150bdfg8+634cdfg8-2988d2fg8+\
   13594befg8+4308cefg8-8890defg8+2683e2fg8-12214bf2g8-15220cf2g8-11140df2g8\
   +13628ef2g8-9408f3g8-11792bcg9+2335c2g9-7757bdg9+14314cdg9-7420d2g9+8688b\
   eg9+15983ceg9-7923deg9+4081e2g9+15292bfg9-5031cfg9-10115dfg9-3189efg9-595\
   f2g9-15987bg10-8567cg10+7861dg10-10029eg10+6751fg10-5681g11-11011c4-1400b\
   d3+1762cd3+9937d4+4209c3e+14093bcde+8268c2de+7236bd2e-9968cd2e+11754d3e-4\
   981bce2+125c2e2-13637bde2+1395cde2-944d2e2-9583be3-4183ce3-10612de3+9359e\
   4-717c3f+9174bcdf+10011c2df+8611bd2f-11395cd2f+1591d3f+13984bcef-7695c2ef\
   +15727bdef+7257cdef-762d2ef+2545be2f+3337ce2f-11086de2f-14095e3f+11914bcf\
   2-11290c2f2-10924bdf2+7600cdf2-10229d2f2+7146bef2+6242cef2+14666def2+3350\
   e2f2+2455bf3-6668cf3-11754df3+15772ef3-11113f4+2938c3g-2207bcdg-3230c2dg-\
   14755bd2g-15298cd2g-7477d3g-15517bceg-934c2eg-15389bdeg-3168cdeg+6383d2eg\
   +14095be2g-4583ce2g+1521de2g-8444e3g+7684bcfg-15893c2fg-4558bdfg+9488cdfg\
   -8274d2fg+2395befg+12299cefg-3103defg+4488e2fg+2462bf2g-15979cf2g+9867df2\
   g-2415ef2g+13013f3g-821bcg2+15983c2g2+5424bdg2-12572cdg2-1905d2g2+3861beg\
   2-12360ceg2-1845deg2-6018e2g2+6313bfg2+10414cfg2+1605dfg2+9080efg2+13769f\
   2g2+10911bg3+8296cg3+9754dg3-4921eg3-15385fg3+15474g4
```

```
↦ j[159]=be2fg7+1246e3g8-4701bcfg8+14938c2fg8+8883bdfg8+15095cdfg8+11291d2f\
  g8+4037befg8+9270cefg8+15305defg8-15255e2fg8-5595bf2g8-8998cf2g8+3418df2g\
  8+7268ef2g8+14085f3g8+12363bcg9-7873c2g9-6270bdg9-5828cdg9-2294d2g9+2163b\
  eg9-710ceg9+8811deg9+3252e2g9-10223bfg9+13356cfg9+5691dfg9-11900efg9-4214\
  f2g9-1185bg10-13941cg10-3843dg10+15634eg10-6830fg10-14257g11+13681c4-8752\
  bd3-8530cd3+1680d4+5824c3e-12085bcde-11431c2de-4942bd2e-2679cd2e-13753d3e\
  +14563bce2-10511c2e2+13162bde2-6992cde2-2660d2e2-7894be3-14385ce3-4776de3\
  -5920e4+15658c3f+14709bcdf+10163c2df-6525bd2f-3722cd2f+2951d3f-8331bcef+6\
  291c2ef+13948bdef+4430cdef+9429d2ef+13847be2f-11294ce2f+15450de2f-3437e3f\
  +6616bcf2+12409c2f2+2211bdf2+4752cdf2+711d2f2+2856bef2-12392cef2-13998def\
  2+447e2f2+1437bf3+15716cf3+2264df3-3539ef3-10926f4+3036c3g-2879bcdg-7687c\
  2dg-14241bd2g-3771cd2g+9894d3g+6518bceg-829c2eg-2281bdeg-1222cdeg+1123d2e\
  g+11748be2g+13017ce2g-10791de2g+8911e3g-11772bcfg-15713c2fg+8724bdfg-1307\
  8cdfg+10068d2fg-4300befg+10114cefg+3813defg+8356e2fg-2448bf2g-2103cf2g-80\
  59df2g+4237ef2g-6715f3g+14239bcg2-8053c2g2+13306bdg2+4191cdg2-3738d2g2+89\
  27beg2-13411ceg2+3865deg2+9816e2g2-10536bfg2+10411cfg2-7267dfg2+14965efg2\
  +11340f2g2+14708bg3-9217cg3-2042dg3-5253eg3+540fg3-373g4
↦ j[160]=d2efg7+9708e3g8-11505bcfg8-8680c2fg8+3689bdfg8-9569cdfg8+2226d2fg8\
  +15760befg8+13913cefg8-2400defg8+4680e2fg8+899bf2g8-11950cf2g8-4269df2g8+\
  10142ef2g8-11799f3g8-6524bcg9+4288c2g9+14937bdg9+10677cdg9+1939d2g9-4618b\
  eg9-3394ceg9+1122deg9-2772e2g9+6786bfg9-7500cfg9+5459dfg9+7516efg9-10948f\
  2g9-12766bg10+12237cg10-15584dg10+12263eg10-11801fg10+6410g11+14632c4+103\
  86bd3+69cd3+13128d4+10414c3e+13552bcde-6095c2de+5538bd2e+453cd2e+10269d3e\
  -10830bce2+4310c2e2+8246bde2+11900cde2+4238d2e2-8072be3+4877ce3+1257de3-1\
  5265e4+4909c3f+14199bcdf-580c2df+15121bd2f-7279cd2f-10366d3f+7882bcef-119\
  5c2ef+14935bdef+3313cdef+9522d2ef-10405be2f+9075ce2f+1181de2f-14205e3f-15\
  061bcf2-5382c2f2-12773bdf2-3037cdf2-2872d2f2-10978bef2+13563cef2-11968def\
  2+382e2f2+6560bf3+1222cf3-42df3-11742ef3-9329f4-14028c3g-10467bcdg-5015c2\
  dg+9109bd2g+3333cd2g+9771d3g-5824bceg+3670c2eg+928bdeg-2141cdeg+13701d2eg\
  +15072be2g-14854ce2g-11573de2g+5532e3g-2211bcfg+7838c2fg-10701bdfg+13758c\
  dfg-13461d2fg-9574befg-772cefg-8968defg-6871e2fg-8817bf2g-1287cf2g-1100df\
  2g+15027ef2g-4273f3g+4180bcg2-15201c2g2-5493bdg2+4726cdg2-6896d2g2+5103be\
  g2-1571ceg2-5025deg2+14018e2g2+4371bfg2+8037cfg2+3398dfg2+460efg2+6688f2g\
  2+11369bg3+12059cg3-8043dg3-8976eg3+881fg3-14114g4
↦ j[161]=cdefg7-7822e3g8-12649bcfg8-3357c2fg8+12703bdfg8-6408cdfg8+14848d2f\
  g8+922befg8-2449cefg8+11693defg8-2038e2fg8+4430bf2g8-6232cf2g8-3131df2g8+\
  2553ef2g8+5253f3g8+6643bcg9-9398c2g9+10584bdg9-9176cdg9+4009d2g9+6471beg9\
  -7845ceg9-1160deg9-1275e2g9-15597bfg9-9723cfg9-6809dfg9-6858efg9+9917f2g9\
  +13061bg10-5862cg10+5263dg10-12524eg10-2654fg10-14338g11-2888c4-7150bd3+1\
  1158cd3+3413d4-15869c3e-6154bcde+8221c2de+6614bd2e+5903cd2e+1614d3e-9256b\
  ce2+11330c2e2+8147bde2-8649cde2-10285d2e2-150be3+9666ce3+13895de3-5084e4+\
  544c3f-14699bcdf-6604c2df+161bd2f+10283cd2f-7122d3f+6229bcef-7498c2ef+571\
  8bdef-3526cdef+12875d2ef+1266be2f-4231ce2f-8861de2f-8893e3f-7414bcf2-3393\
  c2f2-14752bdf2-14131cdf2+3027d2f2-8358bef2+8102cef2+7208def2-4171e2f2+778\
  7bf3-6408cf3+14561df3+11510ef3-12737f4-4191c3g+4591bcdg-6296c2dg+6734bd2g\
  -2121cd2g+4702d3g-12880bceg+5847c2eg-9110bdeg-11197cdeg-1425d2eg-11126be2\
  g-7711ce2g-9193de2g+14757e3g-3947bcfg-8144c2fg-11612bdfg-946cdfg-6327d2fg\
  -1806befg-454cefg+15278defg-1881e2fg+13838bf2g+15991cf2g-10069df2g+13604e\
  f2g+1378f3g+3295bcg2-15250c2g2+8058bdg2-12832cdg2-1728d2g2-8038beg2+6783c\
  eg2-631deg2-390e2g2-10256bfg2+1387cfg2+1035dfg2-9538efg2-6286f2g2+5699bg3\
  -3584cg3-9379dg3-2179eg3-10940fg3-3520g4
↦ j[162]=bdefg7-13002e3g8+15232bcfg8+8452c2fg8-9443bdfg8-12031cdfg8-7219d2f\
  g8+3824befg8-5591cefg8-14234defg8+1193e2fg8-15790bf2g8-10493cf2g8+280df2g\
   8+13613ef2g8-6402f3g8+12637bcg9-513c2g9-11118bdg9-2220cdg9-260d2g9+4645be\
   g9-3673ceg9+9823deg9-11007e2g9+11964bfg9-1280cfg9+15993dfg9+5744efg9-9941\
   f2g9-2862bg10+9342cg10-8734dg10+12420eg10+11011fg10+5889g11+10231c4-711bd\
   3-14005cd3+3174d4+733c3e-7045bcde+5925c2de+9542bd2e+8472cd2e+9091d3e-1745\
   bce2-9430c2e2-921bde2-6949cde2+1305d2e2-12465be3-13410ce3+3889de3+14176e4\
   -12166c3f+2868bcdf+8573c2df+4978bd2f+13350cd2f-5910d3f+5389bcef+3064c2ef+\
   12346bdef+1865cdef-7740d2ef+8601be2f-6966ce2f+6258de2f+8648e3f-1858bcf2-8\
   524c2f2-6770bdf2+12782cdf2-14844d2f2-6016bef2+108cef2-6341def2+14582e2f2+\
   14588bf3-3547cf3-9428df3+5654ef3+15930f4-13853c3g+8253bcdg+4934c2dg+10574\
   bd2g-9443cd2g-9990d3g+6694bceg-12190c2eg-7902bdeg-10752cdeg-11411d2eg+199\
   be2g+13989ce2g-1573de2g-8770e3g+4703bcfg+4675c2fg+1760bdfg-15469cdfg-1500\
   1d2fg+2788befg+1439cefg-9510defg+11903e2fg-7326bf2g-11618cf2g-10066df2g+1\
   1008ef2g-5277f3g-9731bcg2+12694c2g2-9772bdg2+9069cdg2-11273d2g2-9188beg2-\
   2224ceg2-7937deg2+7033e2g2+13514bfg2+13094cfg2+999dfg2+6563efg2+3578f2g2-\
   2434bg3-5684cg3+8546dg3-44eg3-4278fg3-12877g4
↦ j[163]=c2efg7-15127e3g8+14419bcfg8-969c2fg8+10781bdfg8-10092cdfg8+15448d2\
   fg8+12487befg8-15579cefg8-7098defg8-15180e2fg8-2588bf2g8-9721cf2g8+1197df\
   2g8-5857ef2g8-13374f3g8-1855bcg9+811c2g9+12837bdg9+8310cdg9+9562d2g9+4612\
   beg9-13433ceg9+12058deg9+12981e2g9-1336bfg9-11504cfg9-2692dfg9+10874efg9-\
   12359f2g9+967bg10-15946cg10+14392dg10-142eg10+4011fg10-9224g11+14392c4-31\
   80bd3-5462cd3-4378d4+15650c3e+7582bcde-12632c2de-7951bd2e+1947cd2e+4752d3\
   e-13919bce2+5425c2e2+820bde2-13385cde2+10725d2e2-12964be3+10349ce3-12547d\
   e3-1308e4+7654c3f-13060bcdf+4957c2df-14115bd2f+2742cd2f+3514d3f+241bcef+8\
   675c2ef+9221bdef+15808cdef-5985d2ef-1416be2f+10462ce2f+11456de2f-6026e3f-\
   10950bcf2+3915c2f2-4635bdf2+11856cdf2+7221d2f2-7848bef2+2669cef2-9762def2\
   +9125e2f2+10192bf3+2599cf3-8690df3+4368ef3-9305f4+8824c3g+5918bcdg+4458c2\
   dg-14765bd2g-13692cd2g+1173d3g-13629bceg-2769c2eg+8029bdeg+12422cdeg+1070\
   2d2eg+6720be2g-15634ce2g+15933de2g-5799e3g-7368bcfg-1000c2fg+1755bdfg+229\
   7cdfg-11867d2fg+12424befg+7425cefg-11243defg+7768e2fg+9602bf2g-5677cf2g-1\
   4458df2g-2043ef2g-7897f3g-15626bcg2+5574c2g2+12282bdg2-1819cdg2+2769d2g2+\
   13091beg2+4292ceg2+14572deg2-12659e2g2+1737bfg2-915cfg2+9292dfg2+12507efg\
   2-7620f2g2-8512bg3-1933cg3+10625dg3-3610eg3-13923fg3-8204g4
↦ j[164]=bcefg7+7494e3g8-13140bcfg8+15925c2fg8+8558bdfg8+5876cdfg8-11196d2f\
   g8-15067befg8-10681cefg8+420defg8-1110e2fg8+5071bf2g8+6783cf2g8-14593df2g\
   8-6903ef2g8-13324f3g8-12742bcg9+12555c2g9+14734bdg9+3cdg9+11920d2g9+14452\
   beg9-4047ceg9-2758deg9-4906e2g9+10265bfg9-15552cfg9+4776dfg9+3619efg9-102\
   55f2g9+14084bg10-15665cg10-1715dg10+3606eg10-14528fg10+3660g11-9142c4-115\
   07bd3-5740cd3-7533d4+8946c3e+13654bcde-15701c2de-4875bd2e-9641cd2e+12521d\
   3e+12112bce2+9462c2e2-3424bde2-13326cde2+6526d2e2+4029be3-4570ce3-4960de3\
   -995e4-12222c3f+6615bcdf-14784c2df+15447bd2f+2721cd2f+3161d3f-776bcef+891\
   7c2ef-8955bdef+7500cdef+8836d2ef+5198be2f+4386ce2f+6509de2f+13938e3f-2676\
   bcf2-4518c2f2+6750bdf2-6139cdf2-12393d2f2+4221bef2-12976cef2+4176def2-144\
   2e2f2+12545bf3+2867cf3+15544df3-14839ef3-8306f4+4613c3g+4269bcdg-8314c2dg\
   -15299bd2g-4730cd2g-7381d3g-6233bceg+1382c2eg+5752bdeg-15416cdeg-14183d2e\
   g-7142be2g+6431ce2g-6326de2g-14336e3g+13749bcfg+14205c2fg+5434bdfg+6744cd\
   fg-11196d2fg+7239befg+2481cefg-1785defg-15809e2fg+14698bf2g+9494cf2g-1285\
   0df2g-11499ef2g+1202f3g+12856bcg2+4730c2g2+5016bdg2-2000cdg2+6655d2g2+268\
   9beg2+15312ceg2+9812deg2-6250e2g2-10463bfg2+3105cfg2-3696dfg2-10156efg2+1\
   662f2g2-11130bg3+15058cg3-2686dg3-6512eg3-12237fg3-1726g4
↦ j[165]=d3fg7-4997e3g8-1201bcfg8+9152c2fg8+8391bdfg8-15976cdfg8-1491d2fg8+\
   2416befg8-11260cefg8-14605defg8+15201e2fg8-1249bf2g8+13951cf2g8-15542df2g\
   8+8750ef2g8+12298f3g8-14957bcg9-4903c2g9-14363bdg9-3492cdg9-11780d2g9-158\
   15beg9-11413ceg9+12128deg9-12862e2g9+13086bfg9+11076cfg9-13021dfg9+2813ef\
   g9-10880f2g9+13705bg10+6391cg10+13308dg10+14513eg10+4739fg10-13418g11+745\
   c4+15783bd3-7359cd3-12432d4+11164c3e+10727bcde+8211c2de+8026bd2e-8379cd2e\
   +822d3e-11885bce2+4126c2e2-7341bde2+5793cde2-11614d2e2-185be3-8902ce3-131\
   22de3-11846e4+6606c3f+222bcdf-2242c2df+14740bd2f+568cd2f-8489d3f+9020bcef\
   +14145c2ef+14978bdef+7769cdef+5904d2ef-11915be2f-8043ce2f-8745de2f+9276e3\
   f-81bcf2+56c2f2+6621bdf2+12830cdf2-15657d2f2-6011bef2+1287cef2+13326def2+\
   284e2f2+9328bf3+15037cf3-15379df3+15891ef3+12002f4-6040c3g+4340bcdg-7031c\
   2dg-1421bd2g-11677cd2g-7058d3g+2046bceg-13335c2eg+2383bdeg-8777cdeg-7117d\
   2eg-8670be2g-13206ce2g-2583de2g+12692e3g-2385bcfg-5568c2fg-7048bdfg+4652c\
   dfg-11493d2fg-14754befg-12336cefg+7587defg-2045e2fg+13651bf2g+3466cf2g-81\
   09df2g-14910ef2g-11070f3g-9540bcg2+15715c2g2-6319bdg2-5950cdg2+8406d2g2+9\
   655beg2+13191ceg2-1816deg2+5943e2g2+13693bfg2+1390cfg2+7222dfg2+6694efg2-\
   6418f2g2-5668bg3+2854cg3-3358dg3-9449eg3+10743fg3+12471g4
↦ j[166]=cd2fg7-9537e3g8+4261bcfg8-15185c2fg8+3931bdfg8+6300cdfg8+6255d2fg8\
   +2955befg8-5569cefg8+3325defg8+7731e2fg8+12737bf2g8+8657cf2g8+3697df2g8-7\
   707ef2g8+6107f3g8-11511bcg9-4509c2g9-7779bdg9+8853cdg9+9801d2g9-12029beg9\
   +7329ceg9+15219deg9+9595e2g9-8524bfg9-5521cfg9-5804dfg9+7212efg9-4493f2g9\
   -91bg10-13950cg10+2942dg10-14462eg10+13668fg10-6709g11+12214c4+6435bd3-74\
   12cd3-12978d4+9109c3e+5174bcde+5928c2de+8062bd2e-4412cd2e-1145d3e-1137bce\
   2+1232c2e2+4853bde2-7880cde2+4153d2e2-4385be3+7516ce3-12244de3-1550e4-190\
   c3f+7975bcdf+11699c2df+988bd2f+14826cd2f-15077d3f+6871bcef-4071c2ef+4014b\
   def-12099cdef+8486d2ef+627be2f-44ce2f+10403de2f-6048e3f+15284bcf2-7629c2f\
   2+5244bdf2-11016cdf2+7163d2f2-6429bef2+4666cef2-4291def2+2205e2f2+6550bf3\
   -14840cf3+6517df3+13091ef3-9205f4+6106c3g+13409bcdg-10127c2dg+15767bd2g-2\
   792cd2g+6612d3g-5148bceg-3041c2eg+6415bdeg+14157cdeg+9711d2eg+14829be2g-1\
   3622ce2g+14437de2g+6573e3g-3846bcfg+6974c2fg+6730bdfg-13046cdfg-12182d2fg\
   +8656befg+15293cefg+14276defg+2875e2fg+7463bf2g-12337cf2g-11005df2g-5706e\
   f2g+8717f3g-12625bcg2+13271c2g2-8791bdg2-409cdg2-8525d2g2+1557beg2+12750c\
   eg2-12934deg2+3138e2g2-13733bfg2+1926cfg2+15911dfg2+9429efg2+7915f2g2-107\
   53bg3-3408cg3+13864dg3+6161eg3-2043fg3+6767g4
↦ j[167]=bd2fg7+9701e3g8-13865bcfg8-1824c2fg8-14506bdfg8-11770cdfg8-14456d2\
   fg8-9233befg8-771cefg8+4703defg8-2509e2fg8-6745bf2g8-15275cf2g8+11107df2g\
   8+14945ef2g8+3438f3g8+2222bcg9+13160c2g9+9805bdg9+14460cdg9-738d2g9+8899b\
   eg9-9705ceg9+10098deg9+1784e2g9+9896bfg9-1219cfg9+1927dfg9-13227efg9-2589\
   f2g9-9913bg10-6023cg10-12433dg10+2005eg10+87fg10+14445g11-14620c4-15966bd\
   3+428cd3+14582d4-8375c3e-14265bcde-8116c2de+10008bd2e+11485cd2e-12863d3e+\
   5434bce2+3368c2e2+3729bde2-9902cde2-15914d2e2-12122be3+6525ce3+2651de3+12\
   318e4-813c3f-6331bcdf+2997c2df+11200bd2f+13948cd2f+5328d3f+5735bcef+3751c\
   2ef-10849bdef-4564cdef+4268d2ef+1500be2f+14889ce2f-2065de2f-12718e3f-4799\
   bcf2+11895c2f2+1316bdf2-970cdf2+1591d2f2-1144bef2-5114cef2+5124def2-2681e\
   2f2+7532bf3-5977cf3+2071df3+2260ef3+8404f4-11991c3g+10793bcdg-9172c2dg+39\
   4bd2g-1298cd2g-3139d3g-2957bceg+11595c2eg-9261bdeg+10862cdeg-8863d2eg+191\
   0be2g-1867ce2g-8736de2g-2679e3g-3806bcfg+12259c2fg-10451bdfg-4044cdfg+238\
   4d2fg-15300befg-14335cefg+14042defg-10066e2fg+6354bf2g+1634cf2g-14018df2g\
   -67ef2g-3109f3g+9095bcg2-4988c2g2+14755bdg2-4584cdg2-12029d2g2+7715beg2+4\
   547ceg2+6148deg2+12712e2g2-4097bfg2+14905cfg2+696dfg2-3606efg2-5560f2g2+5\
   314bg3+10193cg3+7483dg3-5358eg3+3996fg3+15544g4
↦ j[168]=c2dfg7-2054e3g8-11396bcfg8+14982c2fg8+2063bdfg8-13440cdfg8-13212d2\
   fg8-3336befg8-5241cefg8+8989defg8+12177e2fg8-3943bf2g8-3965cf2g8-14209df2\
   g8+7038ef2g8+55f3g8+14334bcg9-1246c2g9+14589bdg9+11835cdg9-3325d2g9-12376\
   beg9+14056ceg9+14693deg9+8532e2g9-1953bfg9-3053cfg9-10016dfg9+15502efg9-9\
   151f2g9-7330bg10-3270cg10-15336dg10+13994eg10-13137fg10+13643g11-11237c4-\
   6514bd3-7260cd3+12816d4+8788c3e-14500bcde+6005c2de+2186bd2e-2112cd2e-2128\
   d3e+10935bce2+11296c2e2-12546bde2-9203cde2+9210d2e2-2648be3+15413ce3+1438\
   0de3-7590e4-14019c3f-3736bcdf+14673c2df+8316bd2f-3966cd2f-11587d3f+8030bc\
   ef+11866c2ef-8503bdef-6809cdef-3193d2ef-14830be2f-5057ce2f-426de2f-2295e3\
   f+15182bcf2-3229c2f2-15060bdf2+15843cdf2+383d2f2-12346bef2-3159cef2-12603\
   def2+679e2f2+649bf3-4087cf3-4458df3+15873ef3-12869f4+9674c3g+13225bcdg+11\
   688c2dg-15919bd2g+7668cd2g-12707d3g-15984bceg+4132c2eg+6732bdeg+395cdeg+1\
   3236d2eg+440be2g-1725ce2g-4599de2g+15887e3g-15464bcfg-13790c2fg-4482bdfg-\
   4464cdfg-10793d2fg-11367befg-607cefg+1842defg+5738e2fg+10787bf2g-10619cf2\
   g+1310df2g+11506ef2g-15118f3g-7731bcg2-12989c2g2-933bdg2-153cdg2+4951d2g2\
   -6853beg2+8534ceg2-7645deg2+5495e2g2+9328bfg2-13866cfg2+7616dfg2+15247efg\
   2-4912f2g2-3055bg3+5583cg3+2867dg3+8175eg3+12952fg3-11283g4
↦ j[169]=bcdfg7-906e3g8-13357bcfg8+8375c2fg8+3405bdfg8-4675cdfg8+14989d2fg8\
   -8204befg8+1550cefg8+11328defg8-1800e2fg8+10176bf2g8+14537cf2g8+2371df2g8\
   +1977ef2g8+1853f3g8+13878bcg9-11335c2g9+15100bdg9+8502cdg9-9934d2g9+7483b\
   eg9+2669ceg9-14974deg9-12023e2g9+9427bfg9+9666cfg9+6235dfg9+2327efg9+9183\
   f2g9-7262bg10-12414cg10+12800dg10-15123eg10+13547fg10+10532g11-9053c4+292\
   4bd3-13038cd3-461d4-6653c3e+7418bcde-2175c2de-14305bd2e-14846cd2e+7268d3e\
   +9935bce2-5172c2e2-6274bde2-12948cde2-12431d2e2-8383be3+714ce3+9786de3-71\
   53e4-1251c3f-5800bcdf-2706c2df+14491bd2f-9cd2f+14300d3f+6398bcef+1124c2ef\
   +9628bdef-2114cdef-6325d2ef+14565be2f-1910ce2f-2485de2f+12055e3f-6437bcf2\
   +8238c2f2-14417bdf2+3093cdf2-2350d2f2+15235bef2+9144cef2-3937def2+11665e2\
   f2+2846bf3+7896cf3-2410df3-11274ef3-14715f4+13991c3g-2009bcdg+8093c2dg+36\
   bd2g+4781cd2g+14908d3g+7682bceg-8456c2eg-5813bdeg-2598cdeg+14715d2eg+6946\
   be2g-4131ce2g-10499de2g+13669e3g+5165bcfg+9774c2fg-8014bdfg-6160cdfg-6948\
   d2fg+6675befg+6857cefg-6364defg+2079e2fg+15036bf2g-14268cf2g-14386df2g+79\
   82ef2g-4216f3g+14562bcg2+13063c2g2+7192bdg2+5985cdg2+3998d2g2+9793beg2+14\
   549ceg2-11416deg2+3027e2g2+10676bfg2+9967cfg2+3072dfg2+4258efg2-11037f2g2\
   +1614bg3+10805cg3-8197dg3+8921eg3-3786fg3+2850g4
↦ j[170]=c3fg7-1444e3g8+410bcfg8-15070c2fg8+15900bdfg8+12124cdfg8-10663d2fg\
   8+8738befg8+8624cefg8-4867defg8+12235e2fg8+11638bf2g8+4912cf2g8+2726df2g8\
   +7789ef2g8-13423f3g8+12908bcg9-13159c2g9+13189bdg9+9421cdg9-7485d2g9-1768\
   beg9+12286ceg9+11569deg9-1634e2g9-1304bfg9+4cfg9+9436dfg9+9857efg9-12226f\
   2g9+2224bg10+9931cg10+8974dg10+3634eg10+4144fg10-15654g11+5151c4-5811bd3+\
   7714cd3+10472d4+1962c3e-403bcde-354c2de-11085bd2e-196cd2e-9505d3e+5655bce\
   2-15125c2e2+11384bde2+15857cde2-43d2e2-1063be3-15026ce3+1724de3+6070e4+73\
   10c3f+9233bcdf+4267c2df-7391bd2f+9451cd2f+2641d3f-1812bcef+379c2ef-6205bd\
   ef-5722cdef-15899d2ef-3020be2f+9755ce2f+12001de2f-8724e3f+8779bcf2-2298c2\
   f2+478bdf2+1213cdf2-2187d2f2-6877bef2+13502cef2-6661def2+7488e2f2-13918bf\
   3+3710cf3+9701df3-3982ef3+2402f4+8438c3g-4734bcdg+11428c2dg+15739bd2g-195\
   5cd2g+1323d3g+5338bceg+864c2eg-9576bdeg+6684cdeg-15803d2eg+5541be2g+7679c\
   e2g-8424de2g+14870e3g+6161bcfg-10190c2fg+4478bdfg-8740cdfg-11526d2fg-8270\
   befg+3494cefg+2784defg+7804e2fg+8458bf2g-6471cf2g+629df2g+1406ef2g-11802f\
   3g+11211bcg2-4178c2g2-11488bdg2+4601cdg2-15098d2g2+10925beg2+10865ceg2-15\
   004deg2+4634e2g2-12708bfg2+5443cfg2+9611dfg2+3647efg2-1754f2g2+3052bg3+22\
   36cg3-2170dg3+14497eg3+296fg3+6801g4
↦ j[171]=e4g7+5785e3g8-3232bcfg8+11011c2fg8-5705bdfg8-6222cdfg8-10516d2fg8+\
   3651befg8-4085cefg8-10135defg8+13480e2fg8-15438bf2g8-13043cf2g8+9414df2g8\
   -13473ef2g8-164f3g8+8603bcg9+1511c2g9+8617bdg9-13299cdg9+13837d2g9-2020be\
   g9+12546ceg9-14845deg9-11275e2g9-14603bfg9-6310cfg9-13954dfg9+14754efg9+3\
   130f2g9-1864bg10+12126cg10-9282dg10-8245eg10-15112fg10-14730g11+8722c4-11\
   797bd3+1185cd3-58d4-15865c3e-9337bcde+7079c2de-6924bd2e-5232cd2e-10651d3e\
   -1415bce2-3076c2e2+15664bde2+10384cde2-10906d2e2+9203be3+12039ce3+1749de3\
   +15780e4+4934c3f-13117bcdf-9456c2df+12843bd2f+11031cd2f+13259d3f+9327bcef\
   -14217c2ef+245bdef-3929cdef+1654d2ef+15135be2f-5208ce2f+9050de2f-4475e3f-\
   9098bcf2+14789c2f2+7573bdf2+11885cdf2-13756d2f2-6383bef2+7556cef2+15347de\
   f2+6383e2f2+15967bf3+4375cf3-3519df3-7373ef3-3145f4+3067c3g+5591bcdg+2741\
   c2dg-11252bd2g+11990cd2g-6687d3g+14094bceg+713c2eg+6228bdeg+656cdeg-306d2\
   eg+2368be2g-15255ce2g+5855de2g+9439e3g-443bcfg+8602c2fg+8598bdfg-8341cdfg\
   +15202d2fg+1212befg-6056cefg+2217defg-1298e2fg-12989bf2g-6630cf2g+2914df2\
   g-13452ef2g-3454f3g+52bcg2-547c2g2-7808bdg2-15021cdg2-14020d2g2-2007beg2-\
   8652ceg2-2475deg2+12122e2g2-3350bfg2-14124cfg2-13434dfg2+1544efg2+3300f2g\
   2-6189bg3+10002cg3-2608dg3-6099eg3+4002fg3+847g4
↦ j[172]=de3g7-2143e3g8-4685bcfg8-4751c2fg8-9086bdfg8-2478cdfg8-6971d2fg8+1\
   5651befg8+14834cefg8+6400defg8+12741e2fg8-15930bf2g8+13185cf2g8+13075df2g\
   8+12027ef2g8-4378f3g8-5553bcg9-7842c2g9-11929bdg9+11791cdg9+12944d2g9+596\
   5beg9+14564ceg9+9281deg9+6038e2g9-3353bfg9-12121cfg9+2033dfg9-3623efg9-99\
   64f2g9-6341bg10+7067cg10+14506dg10+1800eg10+472fg10-2145g11+2245c4-861bd3\
   +13008cd3+10503d4-6102c3e-14139bcde-2052c2de-10725bd2e+10751cd2e-11501d3e\
   -9530bce2-2184c2e2-7788bde2+7551cde2-9294d2e2+6104be3+15474ce3+14426de3+1\
   2675e4+7073c3f+13153bcdf+1425c2df-14310bd2f+9789cd2f+5213d3f-3460bcef+112\
   73c2ef+1301bdef-2720cdef-7521d2ef+4809be2f-1071ce2f-14046de2f+4734e3f+697\
   1bcf2+7198c2f2+9195bdf2+10749cdf2-4141d2f2+2664bef2-12570cef2-5658def2-97\
   95e2f2+15308bf3-5820cf3+3895df3+10375ef3-513f4+3448c3g-13532bcdg-1467c2dg\
   +5256bd2g+2015cd2g+10011d3g-4641bceg-15192c2eg-14570bdeg+10176cdeg+7548d2\
   eg+15710be2g-1148ce2g-10786de2g-6834e3g-11615bcfg+5553c2fg+3357bdfg-6194c\
   dfg+1246d2fg+1574befg-7792cefg-6414defg-7338e2fg-13258bf2g-4260cf2g+5427d\
   f2g+15968ef2g+7504f3g+13510bcg2+11300c2g2+15577bdg2+11840cdg2-15895d2g2+1\
   4370beg2+13382ceg2-7080deg2-927e2g2+5400bfg2+9700cfg2+5443dfg2+14804efg2-\
   14721f2g2-12572bg3-13766cg3-14641dg3-5532eg3+12246fg3-7321g4
↦ j[173]=ce3g7+3622e3g8+4327bcfg8-15430c2fg8+4235bdfg8-4520cdfg8+8272d2fg8-\
   1371befg8+11682cefg8-15657defg8+4680e2fg8+10604bf2g8-1013cf2g8+12100df2g8\
   -2038ef2g8-4296f3g8+10903bcg9-11061c2g9-10360bdg9+12816cdg9+7863d2g9+1958\
   beg9+5057ceg9-15685deg9-1272e2g9+8759bfg9+8345cfg9+10514dfg9-4698efg9+607\
   8f2g9-9880bg10-1511cg10+4473dg10-10578eg10-8596fg10+7687g11-7868c4-12820b\
   d3-12982cd3-15068d4-10062c3e-2660bcde+2988c2de-9213bd2e-3459cd2e+3814d3e-\
   9763bce2+2123c2e2+9565bde2+5910cde2-5774d2e2+2434be3+14421ce3-9032de3+142\
   04e4+6196c3f+3498bcdf-15753c2df-1764bd2f+10749cd2f-14925d3f+11336bcef-979\
   1c2ef-10760bdef-12442cdef+2651d2ef+4741be2f+10639ce2f-8173de2f-12804e3f+1\
   0930bcf2-14007c2f2+1934bdf2-3770cdf2-3456d2f2+8304bef2+3311cef2+2455def2-\
   15939e2f2+14677bf3+3497cf3+8533df3-3150ef3+12461f4-2293c3g-9176bcdg+7299c\
   2dg-14922bd2g+12622cd2g+13253d3g-3615bceg-10648c2eg-6401bdeg+6768cdeg-141\
   45d2eg-11263be2g+2265ce2g+14271de2g-15097e3g+14203bcfg-9085c2fg-6691bdfg-\
   11260cdfg-9553d2fg+14631befg-15263cefg-14483defg-5114e2fg+14707bf2g-9747c\
   f2g+7802df2g+8872ef2g-15817f3g+5223bcg2-14729c2g2+1609bdg2+13817cdg2-1032\
   4d2g2-14332beg2-6512ceg2+6696deg2+9237e2g2+10208bfg2-11516cfg2+6160dfg2-1\
   0884efg2+10387f2g2-1792bg3-12468cg3+11167dg3-4724eg3+4971fg3+7731g4
↦ j[174]=be3g7+5098e3g8+11267bcfg8+1479c2fg8-1470bdfg8+3559cdfg8-7552d2fg8-\
   5855befg8+14744cefg8+2532defg8+12136e2fg8+3568bf2g8-9513cf2g8+10229df2g8-\
   8502ef2g8-3632f3g8+9668bcg9-10171c2g9-3930bdg9-7180cdg9-13037d2g9+13071be\
   g9+3988ceg9-11929deg9+4482e2g9+8360bfg9+3978cfg9-12758dfg9+2547efg9-1132f\
   2g9-7560bg10-8591cg10-15231dg10+7874eg10+3883fg10+6015g11-7400c4-3872bd3-\
   5484cd3-3821d4+9372c3e+5683bcde-12954c2de+1684bd2e+8875cd2e+1925d3e+13723\
   bce2-4206c2e2+5680bde2-11429cde2-3397d2e2-4544be3-15678ce3+3558de3-4192e4\
   +279c3f-3913bcdf+15652c2df-4736bd2f+8690cd2f-6923d3f+1062bcef+1941c2ef+12\
   986bdef+9115cdef-5291d2ef-1833be2f-10415ce2f+4024de2f-8040e3f+15759bcf2+6\
   546c2f2+2209bdf2-14704cdf2+1092d2f2+1381bef2-9937cef2-10698def2-732e2f2+3\
     525bf3+13217cf3+9014df3+8848ef3+14620f4+7239c3g-5517bcdg-14672c2dg-6687bd\
     2g+3728cd2g-3874d3g-9589bceg-7382c2eg-3446bdeg-4197cdeg-181d2eg-741be2g+1\
     0352ce2g-10701de2g+8463e3g-11437bcfg+6873c2fg-15528bdfg+6635cdfg+9336d2fg\
     +6662befg-6651cefg+2629defg-8036e2fg-4232bf2g+15048cf2g+7267df2g+8118ef2g\
     +13755f3g-10425bcg2-7289c2g2-14282bdg2-11845cdg2-10707d2g2-10367beg2-1061\
     8ceg2+9679deg2-8293e2g2-3860bfg2+1631cfg2+11288dfg2-13616efg2+15507f2g2-1\
     4468bg3+5255cg3+1405dg3-3166eg3-15898fg3+6132g4
↦ j[175]=d2e2g7+4446e3g8-10198bcfg8-8274c2fg8+181bdfg8+4317cdfg8+8671d2fg8-\
     11813befg8+12574cefg8+5742defg8+5786e2fg8-3254bf2g8-9809cf2g8+2038df2g8+1\
     4867ef2g8+9067f3g8+11808bcg9-3654c2g9+1439bdg9-5989cdg9+2584d2g9+8156beg9\
     -12160ceg9+13964deg9+14251e2g9+3053bfg9+3506cfg9+14126dfg9+2624efg9-15206\
     f2g9+9424bg10-11783cg10+8229dg10-9528eg10-11185fg10-12352g11-3934c4-4388b\
     d3-13324cd3-14438d4+13334c3e+9451bcde+5319c2de-1940bd2e+538cd2e+1952d3e+5\
     823bce2-407c2e2-9878bde2+13312cde2-8669d2e2+15342be3-123ce3-1059de3+8779e\
     4-15955c3f+7432bcdf-12403c2df+1532bd2f+13507cd2f-2564d3f-223bcef+15234c2e\
     f+5680bdef+6046cdef-10868d2ef+10495be2f+14649ce2f-9360de2f-1508e3f-13616b\
     cf2-12951c2f2-5485bdf2-11637cdf2+4162d2f2-9524bef2-7800cef2-1048def2-2786\
     e2f2+8492bf3+4537cf3+14620df3-5909ef3-7362f4-10463c3g-485bcdg-890c2dg+108\
     04bd2g-9068cd2g+12245d3g-9247bceg-3061c2eg+2304bdeg-6905cdeg-10315d2eg+47\
     88be2g+14657ce2g+8981de2g-12474e3g-389bcfg+1783c2fg+3281bdfg-13823cdfg-12\
     879d2fg-4170befg+4510cefg+15885defg-4881e2fg-484bf2g-12458cf2g+2241df2g-3\
     715ef2g+11513f3g+12294bcg2-5708c2g2+14470bdg2+6326cdg2-9137d2g2+6006beg2+\
     4124ceg2-7385deg2+12508e2g2-10988bfg2-12393cfg2+6450dfg2-4900efg2+6308f2g\
     2+7833bg3+4589cg3+7129dg3+9598eg3-10491fg3-8641g4
↦ j[176]=g12-3257cde2f-15983d2e2f-6458be3f+6891ce3f-6106de3f-1130e4f+10403c\
     3f2+7302bcdf2-8459c2df2+13265bd2f2+13401cd2f2-8553d3f2+8474bcef2-15850c2e\
     f2-4166bdef2-8399cdef2+9027d2ef2-10be2f2-13194ce2f2+2534de2f2+10989e3f2-1\
     4531bcf3+10285c2f3+7287bdf3+14858cdf3-7457d2f3+11365bef3+3050cef3+11681de\
     f3+14788e2f3-10935bf4-14492cf4-11546df4+4317ef4-8434f5+8193c4g-2675bd3g-1\
     4596cd3g-14442d4g+14541c3eg+8711bcdeg-7076c2deg-1638bd2eg+4244cd2eg-15383\
     d3eg+3574bce2g-9296c2e2g+612bde2g+15506cde2g-10794d2e2g+10494be3g-9872ce3\
     g-432de3g+10103e4g-3879c3fg-9590bcdfg+14202c2dfg-6990bd2fg-3394cd2fg+2044\
     d3fg+2440bcefg+15146c2efg+13bdefg+8485cdefg+13048d2efg-11562be2fg+13648ce\
     2fg-4558de2fg+7764e3fg+7028bcf2g-4029c2f2g+6061bdf2g+11303cdf2g+9302d2f2g\
     +1667bef2g-6719cef2g+15403def2g+11803e2f2g-11267bf3g+4382cf3g+3373df3g+97\
     88ef3g+13244f4g+8166c3g2+8118bcdg2-3980c2dg2+12766bd2g2-2200cd2g2+2155d3g\
     2+3579bceg2+7845c2eg2+1158bdeg2-15690cdeg2-6955d2eg2+3075be2g2-5436ce2g2-\
     11611de2g2-15489e3g2+14154bcfg2-673c2fg2-7535bdfg2+14921cdfg2-787d2fg2-50\
     3befg2+1860cefg2-15250defg2+13227e2fg2-2182bf2g2-14808cf2g2-10257df2g2+29\
     08ef2g2-5957f3g2-8999bcg3-14791c2g3+7998bdg3-6058cdg3-9070d2g3+13651beg3+\
     3961ceg3-1234deg3-7406e2g3-12918bfg3-2456cfg3-13096dfg3+5775efg3-14851f2g\
     3-8830bg4+3060cg4+14945dg4+9029eg4-273fg4+59g5
↦ j[177]=fg11+5602cde2f-11890d2e2f+15224be3f-12933ce3f-3700de3f+7888e4f-111\
     13c3f2-12603bcdf2-8106c2df2-4977bd2f2+6816cd2f2+10592d3f2+6962bcef2-9114c\
     2ef2+4860bdef2+5693cdef2+11658d2ef2+3793be2f2-182ce2f2-9467de2f2-12735e3f\
     2-3857bcf3+4691c2f3-11305bdf3-8043cdf3-10299d2f3-13746bef3-7602cef3-11980\
     def3+4719e2f3-3645bf4+6590cf4+11345df4-7934ef4+9584f5+10504c4g+9855bd3g+5\
     650cd3g+6768d4g+2172c3eg-1740bcdeg-9552c2deg-9784bd2eg-10770cd2eg-6150d3e\
     g-14658bce2g-3219c2e2g-8875bde2g-758cde2g-3883d2e2g-7973be3g-3920ce3g-125\
     31de3g+5544e4g-15634c3fg+6203bcdfg-3863c2dfg+10747bd2fg-6060cd2fg-5895d3f\
     g+8306bcefg+14884c2efg-5505bdefg+8645cdefg-14130d2efg-11158be2fg+13739ce2\
     fg+15260de2fg-13510e3fg+10860bcf2g-15083c2f2g-6298bdf2g-12346cdf2g-12736d\
     2f2g-12834bef2g+14255cef2g-5440def2g-9491e2f2g-10776bf3g+106cf3g-9015df3g\
     -11370ef3g-505f4g+6610c3g2-5824bcdg2-3139c2dg2-14571bd2g2-9720cd2g2-3263d\
     3g2+15649bceg2-8507c2eg2-11763bdeg2-562cdeg2+5452d2eg2-2054be2g2-5021ce2g\
     2+6733de2g2+827e3g2-12432bcfg2+1010c2fg2-548bdfg2-1224cdfg2+15580d2fg2+25\
     68befg2-1836cefg2+11424defg2+6815e2fg2+6182bf2g2-13328cf2g2+751df2g2+1133\
     3ef2g2-11573f3g2-9333bcg3+1615c2g3+14163bdg3+14367cdg3+5469d2g3-13144beg3\
     +6395ceg3-14280deg3+2884e2g3-4457bfg3-13390cfg3-6961dfg3-361efg3+15728f2g\
     3+2479bg4+7371cg4+9952dg4-14261eg4-10976fg4-11673g5
  ↦ j[178]=eg11-15587cde2f+13430d2e2f+9844be3f+3029ce3f-4882de3f-14025e4f-401\
     6c3f2+2783bcdf2-6224c2df2+5825bd2f2+376cd2f2-4988d3f2+12161bcef2+11491c2e\
     f2-8178bdef2+7394cdef2+4215d2ef2-1836be2f2-15220ce2f2+11875de2f2+11647e3f\
     2-3503bcf3-9699c2f3-15073bdf3+8567cdf3-5640d2f3+5242bef3-8864cef3-5173def\
     3-7851e2f3+1358bf4+3847cf4-8678df4+14939ef4+12940f5-5785c4g+4763bd3g+9925\
     cd3g+7896d4g-14327c3eg-2771bcdeg-4577c2deg+6647bd2eg-14664cd2eg+10619d3eg\
     -12384bce2g+5623c2e2g+11634bde2g+2949cde2g-15750d2e2g+7402be3g+6733ce3g-4\
     695de3g+14692e4g-6779c3fg+2415bcdfg-13378c2dfg+6314bd2fg-6905cd2fg-10938d\
     3fg-6245bcefg-819c2efg-14783bdefg-1560cdefg-5935d2efg-11567be2fg+7395ce2f\
     g+11084de2fg-9641e3fg+13662bcf2g-10466c2f2g+7895bdf2g+744cdf2g+12062d2f2g\
     +1416bef2g-13730cef2g-7195def2g-3738e2f2g+3655bf3g-10116cf3g-8380df3g-792\
     2ef3g+13440f4g+3017c3g2+8534bcdg2+4330c2dg2-15411bd2g2-13368cd2g2+13358d3\
     g2-7447bceg2-5184c2eg2-6243bdeg2-5119cdeg2-4373d2eg2+11783be2g2+5056ce2g2\
     -14833de2g2-5573e3g2-6288bcfg2-9157c2fg2-6793bdfg2-15961cdfg2-12389d2fg2+\
     14197befg2+11218cefg2-1582defg2-12905e2fg2+5205bf2g2-4492cf2g2+3365df2g2+\
     2741ef2g2-4167f3g2+3117bcg3+5368c2g3+9629bdg3+13492cdg3+2874d2g3-4824beg3\
     -10803ceg3-1329deg3+1604e2g3+9746bfg3+12555cfg3-523dfg3-11526efg3+5460f2g\
     3-9878bg4-3116cg4-11502dg4+9506eg4+12006fg4-4826g5
  ↦ j[179]=dg11-4180cde2f+2770d2e2f-5543be3f-14850ce3f+8296de3f+152e4f+12947c\
     3f2+9731bcdf2+11300c2df2+6582bd2f2+12831cd2f2-5491d3f2-2173bcef2+206c2ef2\
     +14124bdef2+13946cdef2+5058d2ef2+10864be2f2+7163ce2f2+2154de2f2-12394e3f2\
     +7863bcf3-7457c2f3-5307bdf3-9461cdf3+10188d2f3+5828bef3-7037cef3-7608def3\
     +691e2f3+9478bf4+14884cf4+1714df4-7091ef4-5053f5-14855c4g+10936bd3g-3810c\
     d3g-5474d4g+13098c3eg+4107bcdeg-9769c2deg+5674bd2eg-3893cd2eg+15157d3eg-5\
     851bce2g-408c2e2g+2669bde2g+5297cde2g-12096d2e2g-5714be3g-1808ce3g+4532de\
     3g+8604e4g+9149c3fg+9630bcdfg+5606c2dfg+4793bd2fg+5137cd2fg-15977d3fg-8bc\
     efg-1054c2efg-4413bdefg+246cdefg+5647d2efg+3250be2fg+5885ce2fg-9649de2fg+\
     548e3fg-270bcf2g-4387c2f2g-15082bdf2g-5101cdf2g+6849d2f2g+12538bef2g-2647\
     cef2g+11995def2g-13333e2f2g+11321bf3g+1974cf3g-13996df3g+4481ef3g+7101f4g\
     -7022c3g2+6426bcdg2-5006c2dg2-9664bd2g2+15937cd2g2+5371d3g2+13652bceg2+59\
     6c2eg2-9843bdeg2+7387cdeg2+4549d2eg2+12398be2g2+2773ce2g2-9958de2g2-337e3\
     g2-4292bcfg2+6909c2fg2+10620bdfg2-14924cdfg2-7674d2fg2+1757befg2-7819cefg\
     2-12577defg2+9566e2fg2-1837bf2g2+11361cf2g2-7123df2g2+499ef2g2+13725f3g2+\
     5330bcg3+9355c2g3+15834bdg3-4190cdg3-7098d2g3-12496beg3-224ceg3+10365deg3\
     -9872e2g3-9440bfg3-760cfg3-14293dfg3-5864efg3+11002f2g3-7463bg4+8968cg4-1\
     3293dg4+13411eg4+3299fg4+8236g5
  ↦ j[180]=cg11+13711cde2f-103d2e2f-9261be3f-5141ce3f-12148de3f+3069e4f+3713c\
     3f2+7067bcdf2-12214c2df2+6813bd2f2-7505cd2f2+4312d3f2+10000bcef2+10425c2e\
     f2+5479bdef2-990cdef2+2868d2ef2+13713be2f2+4679ce2f2-2018de2f2-736e3f2+11\
     686bcf3+3388c2f3-5814bdf3+15370cdf3-3093d2f3+365bef3-12329cef3+8661def3+3\
     867e2f3+13196bf4+3084cf4-3217df4+10441ef4+11066f5-9934c4g-15870bd3g-5406c\
     d3g-659d4g-14178c3eg+3104bcdeg-4758c2deg+6574bd2eg-9349cd2eg-1913d3eg-152\
     37bce2g-15710c2e2g-406bde2g+3295cde2g-1387d2e2g-13497be3g+15361ce3g-3653d\
     e3g+13080e4g+15333c3fg-11964bcdfg+6703c2dfg+4395bd2fg-3728cd2fg-704d3fg-4\
     408bcefg-2873c2efg-3916bdefg-568cdefg-9168d2efg+1604be2fg-1137ce2fg-9358d\
     e2fg-4141e3fg-4030bcf2g+6733c2f2g+3252bdf2g-5365cdf2g+14803d2f2g-1444bef2\
     g+4979cef2g+12913def2g+3217e2f2g+4350bf3g+13952cf3g-6305df3g-9138ef3g-172\
     8f4g+3141c3g2-581bcdg2-11805c2dg2+4597bd2g2-13201cd2g2+13813d3g2-10102bce\
     g2+3035c2eg2-4970bdeg2-3508cdeg2-3518d2eg2-6938be2g2+6144ce2g2+13374de2g2\
     +3475e3g2-8031bcfg2+570c2fg2+8338bdfg2+14938cdfg2-5856d2fg2+864befg2-6544\
     cefg2-7097defg2-9396e2fg2+6258bf2g2-4280cf2g2-11069df2g2+13421ef2g2-5052f\
     3g2-9613bcg3-8908c2g3-12111bdg3+2006cdg3+12006d2g3+7061beg3-2831ceg3-4026\
     deg3-1222e2g3+2233bfg3-2342cfg3-5120dfg3-958efg3+1222f2g3-14759bg4-2748cg\
     4-6248dg4-4011eg4-4269fg4+8494g5
  ↦ j[181]=bg11-5655cde2f+5532d2e2f-9000be3f-12426ce3f-9841de3f+13305e4f+1195\
     7c3f2-10384bcdf2-8167c2df2-7412bd2f2-13743cd2f2+7949d3f2-7011bcef2+11851c\
     2ef2-1582bdef2-11438cdef2-235d2ef2-13340be2f2-55ce2f2+5326de2f2+3199e3f2+\
     1549bcf3-15795c2f3+10443bdf3-12465cdf3+9727d2f3+13887bef3-14950cef3-15853\
     def3+15946e2f3-364bf4+12474cf4+10151df4-4028ef4+14639f5+15480c4g-12620bd3\
     g+5583cd3g-11883d4g+3235c3eg+14899bcdeg+5381c2deg+7643bd2eg+5740cd2eg+360\
     4d3eg-10632bce2g-8477c2e2g+6010bde2g-1504cde2g-14255d2e2g-10909be3g-2356c\
     e3g-10257de3g+6431e4g-1259c3fg+1583bcdfg-239c2dfg+12503bd2fg+533cd2fg+688\
     8d3fg+3879bcefg+9242c2efg-3381bdefg-12378cdefg-4181d2efg-5623be2fg-1647ce\
     2fg-15751de2fg-6459e3fg-3965bcf2g+4891c2f2g+11041bdf2g-10326cdf2g-232d2f2\
     g+14407bef2g-6249cef2g-13190def2g+283e2f2g-11780bf3g-10662cf3g+14488df3g-\
     4776ef3g+13826f4g-11102c3g2-14237bcdg2-11468c2dg2-14116bd2g2+7756cd2g2+83\
     31d3g2+5321bceg2+897c2eg2-6570bdeg2+10489cdeg2-7745d2eg2+6013be2g2-15453c\
     e2g2+14533de2g2-12138e3g2-3433bcfg2+11802c2fg2+15671bdfg2-15218cdfg2-1011\
     d2fg2-5489befg2-15366cefg2+12204defg2+4196e2fg2+11627bf2g2-4128cf2g2+1265\
     2df2g2-10423ef2g2-15133f3g2+13521bcg3+12926c2g3+5640bdg3-15901cdg3-12854d\
     2g3+13718beg3+6108ceg3+6632deg3-12743e2g3+14356bfg3+903cfg3+3512dfg3-9282\
     efg3-13744f2g3+139bg4-2460cg4+14263dg4-4597eg4-10906fg4-10473g5
  ↦ j[182]=f2g10-288cde2f-9668d2e2f-9632be3f-6993ce3f-2452de3f-14525e4f+4090c\
     3f2+15029bcdf2-14961c2df2-6066bd2f2+1215cd2f2-3674d3f2-13667bcef2-4437c2e\
     f2+13112bdef2+1161cdef2+10234d2ef2+11139be2f2-10492ce2f2+7818de2f2+8953e3\
     f2+425bcf3+5877c2f3+12517bdf3-15406cdf3+1656d2f3+15762bef3+1920cef3-13745\
     def3-5297e2f3-3226bf4-7402cf4-13504df4-7838ef4-4856f5+3656c4g+5505bd3g-10\
     675cd3g-8546d4g+3856c3eg-12647bcdeg+3284c2deg-4651bd2eg-4486cd2eg+15173d3\
     eg-7416bce2g+4078c2e2g+13145bde2g+14137cde2g+1754d2e2g+7251be3g+10651ce3g\
     -15767de3g-7695e4g-5874c3fg-1152bcdfg+3856c2dfg+4635bd2fg+11325cd2fg-2274\
     d3fg-15635bcefg+4783c2efg-2130bdefg+10156cdefg-14971d2efg+6173be2fg-375ce\
     2fg-9066de2fg-13621e3fg-12003bcf2g+12345c2f2g-12479bdf2g+12115cdf2g+9951d\
     2f2g+8805bef2g+3641cef2g+1354def2g-2815e2f2g+6350bf3g-1417cf3g+9166df3g-1\
     5015ef3g-2184f4g+10993c3g2+8247bcdg2-7967c2dg2+15557bd2g2-1630cd2g2-14511\
     d3g2+14981bceg2-9945c2eg2+7635bdeg2+5026cdeg2-12983d2eg2+1659be2g2-10066c\
     e2g2+766de2g2-413e3g2-4117bcfg2+6726c2fg2+10428bdfg2-15308cdfg2+3584d2fg2\
     +7600befg2+3707cefg2-3845defg2+8971e2fg2+5879bf2g2-2159cf2g2+1624df2g2+62\
     55ef2g2+13433f3g2-8122bcg3+3430c2g3-899bdg3+1032cdg3-3723d2g3+8308beg3-66\
     09ceg3-4054deg3+9137e2g3+15408bfg3+2347cfg3+10178dfg3+12225efg3-15120f2g3\
     -5699bg4+1157cg4-9570dg4-5196eg4+5236fg4-6624g5
  ↦ j[183]=efg10-14268cde2f-12497d2e2f-7654be3f-1037ce3f-8496de3f+4013e4f+466\
     4c3f2-15384bcdf2+4107c2df2-3405bd2f2+7707cd2f2+1455d3f2-2123bcef2+14237c2\
     ef2-7992bdef2+1818cdef2+6574d2ef2+12293be2f2-804ce2f2-7496de2f2-7106e3f2-\
     6281bcf3-9387c2f3+3644bdf3+7022cdf3+5993d2f3-254bef3-553cef3+7408def3-408\
     1e2f3+15630bf4+7526cf4+13741df4-3589ef4+8666f5-8325c4g-272bd3g-14440cd3g+\
     3397d4g+4032c3eg+5264bcdeg+6837c2deg+10782bd2eg+15862cd2eg+58d3eg-13387bc\
     e2g+69c2e2g+13141bde2g+1451cde2g-8903d2e2g-10441be3g+1096ce3g-3962de3g-49\
     29e4g-9291c3fg-6898bcdfg-8113c2dfg-6680bd2fg-4397cd2fg-13911d3fg+5845bcef\
     g-15869c2efg+12659bdefg+8291cdefg-13146d2efg-8107be2fg-38ce2fg+13317de2fg\
     +6538e3fg+2406bcf2g+8797c2f2g-9823bdf2g+4035cdf2g-4228d2f2g-14525bef2g-20\
     49cef2g+9050def2g+13827e2f2g+8777bf3g-5585cf3g+6477df3g-13413ef3g+8201f4g\
     -4213c3g2+6126bcdg2+13952c2dg2+15249bd2g2-6059cd2g2-8724d3g2-3618bceg2-88\
     52c2eg2-7692bdeg2-14040cdeg2-10809d2eg2-2402be2g2-10799ce2g2+1828de2g2+44\
     61e3g2+15168bcfg2+12834c2fg2-8132bdfg2-11723cdfg2+8367d2fg2-10925befg2-39\
     15cefg2-7696defg2-7607e2fg2-5719bf2g2+14062cf2g2-9274df2g2+5189ef2g2-1247\
     f3g2+11461bcg3-777c2g3+1081bdg3+5657cdg3-8822d2g3+11303beg3-2584ceg3+1180\
     1deg3+8887e2g3+11682bfg3+9441cfg3+8982dfg3-8010efg3+7548f2g3-10797bg4-699\
     5cg4+3206dg4-12174eg4-6767fg4-3485g5
  ↦ j[184]=dfg10+13218cde2f-5565d2e2f-10597be3f+5507ce3f-15072de3f+3345e4f+86\
     94c3f2+6071bcdf2-2804c2df2-15778bd2f2+5506cd2f2-710d3f2+6261bcef2-8624c2e\
     f2-13502bdef2+10113cdef2+11274d2ef2-10483be2f2+4775ce2f2+14953de2f2-6794e\
     3f2+6543bcf3-2370c2f3-14919bdf3-5711cdf3+11757d2f3+2940bef3+3024cef3-1463\
     9def3-4134e2f3-464bf4+12987cf4-2396df4-9114ef4-6668f5-1297c4g+12225bd3g-1\
     1095cd3g-1650d4g+3479c3eg+6422bcdeg+5227c2deg-12558bd2eg-8249cd2eg+5376d3\
     eg+13147bce2g+5548c2e2g-1581bde2g+13141cde2g-280d2e2g-4311be3g+5362ce3g-9\
     827de3g-749e4g+10665c3fg+6116bcdfg+4406c2dfg+8191bd2fg-3666cd2fg-12513d3f\
     g+8509bcefg-11467c2efg+9484bdefg+13555cdefg+10673d2efg+4270be2fg+1816ce2f\
     g-14853de2fg+2738e3fg+1091bcf2g-15809c2f2g-15965bdf2g+9405cdf2g-11165d2f2\
     g+11174bef2g-8547cef2g-12810def2g-9508e2f2g+11659bf3g+14993cf3g-12755df3g\
     +11971ef3g+2544f4g+15464c3g2-498bcdg2-3272c2dg2-4778bd2g2-15404cd2g2-1323\
     6d3g2+6642bceg2-10952c2eg2-12714bdeg2-1407cdeg2-3299d2eg2-9551be2g2+6380c\
     e2g2-13728de2g2-9614e3g2+83bcfg2-11021c2fg2+3725bdfg2+11558cdfg2+6575d2fg\
     2+10263befg2+5643cefg2+7043defg2-8196e2fg2-5419bf2g2+2730cf2g2-1175df2g2-\
     12173ef2g2-14225f3g2-11689bcg3-9043c2g3+4985bdg3+11368cdg3+7576d2g3-10043\
     beg3-6688ceg3-12728deg3-12994e2g3-854bfg3+10212cfg3+15197dfg3-84efg3+1304\
     2f2g3+15903bg4-14391cg4+12464dg4+7581eg4+11257fg4+2325g5
  ↦ j[185]=cfg10-14195cde2f+706d2e2f+9078be3f-11099ce3f-12380de3f+13844e4f+10\
     827c3f2+4817bcdf2-10764c2df2+12366bd2f2-11114cd2f2+10432d3f2+8598bcef2+80\
     2c2ef2-4620bdef2+8159cdef2-9716d2ef2+12765be2f2-4713ce2f2+13013de2f2-8290\
     e3f2-561bcf3+10169c2f3-10606bdf3+630cdf3+4633d2f3-2609bef3+3807cef3+6745d\
     ef3-2203e2f3+6692bf4+15890cf4+15152df4+5633ef4+9701f5-1344c4g+1393bd3g+37\
     86cd3g-6441d4g-6546c3eg+14038bcdeg-9535c2deg-4863bd2eg+277cd2eg+12770d3eg\
     -3854bce2g-2078c2e2g+10053bde2g-4230cde2g+12304d2e2g-7301be3g-1505ce3g-14\
     278de3g-258e4g+4759c3fg+5863bcdfg+12324c2dfg+14948bd2fg+11944cd2fg+14805d\
     3fg+11434bcefg+7037c2efg-4931bdefg+4222cdefg-7483d2efg-6513be2fg-14680ce2\
     fg+7291de2fg+10961e3fg+14729bcf2g-1925c2f2g-6262bdf2g-12243cdf2g-8975d2f2\
     g-14418bef2g+13915cef2g+3059def2g+14280e2f2g+11057bf3g-4080cf3g-7835df3g-\
     5350ef3g-5801f4g-9694c3g2+1862bcdg2+14481c2dg2+13559bd2g2+1634cd2g2-11348\
     d3g2+11695bceg2+774c2eg2+4474bdeg2-7330cdeg2+5796d2eg2-2208be2g2-1014ce2g\
     2-8479de2g2-7450e3g2-13625bcfg2+2145c2fg2+1497bdfg2-1913cdfg2-15424d2fg2-\
     8079befg2+4816cefg2-3424defg2+6479e2fg2-13620bf2g2-14800cf2g2+5674df2g2+4\
     63ef2g2-4033f3g2+7802bcg3-374c2g3+13881bdg3-1722cdg3+536d2g3+7309beg3-131\
     3ceg3+15910deg3+8869e2g3+8017bfg3-4639cfg3+15069dfg3-9397efg3+11183f2g3-7\
     848bg4-12870cg4+1832dg4-11287eg4-13669fg4+11705g5
  ↦ j[186]=bfg10-12835cde2f-9832d2e2f+12946be3f-2461ce3f+4267de3f+14871e4f-10\
     830c3f2+11350bcdf2-14690c2df2+3103bd2f2+7535cd2f2+11715d3f2-8870bcef2+404\
     9c2ef2-14924bdef2+12022cdef2-13003d2ef2+9162be2f2-8649ce2f2-8441de2f2-463\
     2e3f2-14759bcf3+4827c2f3+13022bdf3-391cdf3-7955d2f3-12239bef3-4120cef3-60\
     85def3-8777e2f3+12633bf4+12538cf4+7998df4-13922ef4+8960f5+13656c4g+7414bd\
     3g-2672cd3g-14941d4g-1425c3eg-1373bcdeg-13999c2deg-13757bd2eg-15463cd2eg+\
     12287d3eg-1612bce2g+4815c2e2g-5405bde2g+6993cde2g-11390d2e2g+12196be3g+14\
     849ce3g+6595de3g+4564e4g+6013c3fg-7385bcdfg+3307c2dfg-5094bd2fg-15301cd2f\
     g-13286d3fg+2185bcefg+7945c2efg+15712bdefg+12165cdefg-6512d2efg-3083be2fg\
     -3040ce2fg+10987de2fg+6069e3fg+756bcf2g+13195c2f2g+14808bdf2g-14601cdf2g+\
     9916d2f2g-987bef2g-11311cef2g-6912def2g+15029e2f2g+6792bf3g-10114cf3g-586\
     4df3g-9449ef3g+7629f4g-14986c3g2-13215bcdg2-10830c2dg2+13163bd2g2-13928cd\
     2g2+13727d3g2+4530bceg2-10172c2eg2-9466bdeg2+7759cdeg2-14099d2eg2+14674be\
     2g2-8169ce2g2+13052de2g2-7559e3g2+15052bcfg2+2197c2fg2-1244bdfg2-7384cdfg\
     2+5698d2fg2+3245befg2-5916cefg2-1731defg2+13182e2fg2+9861bf2g2-2065cf2g2-\
     8590df2g2-10297ef2g2-4878f3g2+13737bcg3+3753c2g3+5134bdg3+4937cdg3+9502d2\
     g3+8514beg3-379ceg3-13813deg3-7860e2g3-3149bfg3-12095cfg3+9232dfg3-18efg3\
     +13146f2g3+2581bg4+11399cg4-5836dg4-2089eg4-5249fg4+2457g5
  ↦ j[187]=e2g10-5439cde2f+1817d2e2f-2680be3f-587ce3f+15569de3f-4506e4f+3438c\
     3f2+8194bcdf2+7476c2df2-6509bd2f2-13327cd2f2-12432d3f2-3600bcef2+10200c2e\
     f2+201bdef2+13604cdef2-2070d2ef2+10528be2f2-3850ce2f2-11055de2f2+2033e3f2\
     +9608bcf3-7136c2f3-11267bdf3-4934cdf3+11895d2f3+7480bef3+1820cef3-13047de\
     f3+7946e2f3+12612bf4+5527cf4+5179df4-11931ef4-670f5+10161c4g-12272bd3g+93\
     53cd3g+1381d4g-7048c3eg-8592bcdeg-12028c2deg-1814bd2eg-9644cd2eg-2591d3eg\
     -1314bce2g+12876c2e2g+9697bde2g+2721cde2g+14034d2e2g-15262be3g-13518ce3g-\
     12473de3g-3820e4g-5996c3fg+13727bcdfg+8142c2dfg+193bd2fg-2477cd2fg+14765d\
     3fg+12045bcefg+1651c2efg-9558bdefg+13713cdefg+12579d2efg-7343be2fg+15663c\
     e2fg-14869de2fg+5459e3fg-9397bcf2g+1015c2f2g+7294bdf2g-13175cdf2g+4504d2f\
     2g-1829bef2g-6625cef2g+9459def2g-12786e2f2g-569bf3g-4682cf3g+12862df3g-48\
     7ef3g+391f4g+596c3g2+3576bcdg2-1250c2dg2-2730bd2g2-15034cd2g2+7423d3g2-42\
     82bceg2-14720c2eg2+14716bdeg2-862cdeg2-7390d2eg2+6250be2g2-12505ce2g2-214\
     5de2g2-6189e3g2-2602bcfg2+12702c2fg2+7209bdfg2-2993cdfg2-13895d2fg2-12655\
     befg2+115cefg2+10469defg2+9050e2fg2+6620bf2g2+386cf2g2+8526df2g2-2307ef2g\
     2-9255f3g2-14727bcg3+472c2g3-8008bdg3-4984cdg3-2915d2g3-14512beg3+8707ceg\
     3+12247deg3+12933e2g3+2102bfg3+10448cfg3-1930dfg3-7883efg3-7330f2g3-12533\
     bg4+7557cg4-7377dg4-354eg4+5939fg4-10283g5
  ↦ j[188]=deg10-4710cde2f-14675d2e2f-6578be3f+6001ce3f+2285de3f+1638e4f-2115\
     c3f2+1790bcdf2+5070c2df2+6871bd2f2+14993cd2f2+12338d3f2+9406bcef2-13930c2\
     ef2+10084bdef2-5459cdef2-7789d2ef2+11869be2f2+13879ce2f2-9492de2f2-1844e3\
     f2-2899bcf3-14232c2f3-1852bdf3-6140cdf3-10538d2f3+6710bef3+7012cef3+845de\
     f3-13063e2f3+14031bf4-14490cf4+9362df4-233ef4-14396f5+9272c4g-7415bd3g-12\
     939cd3g+13511d4g-15674c3eg+12807bcdeg-4410c2deg-2868bd2eg+14557cd2eg+8277\
     d3eg+10874bce2g-13335c2e2g-5174bde2g+4710cde2g+3800d2e2g+1019be3g+3862ce3\
     g+15774de3g+4685e4g-14900c3fg+14587bcdfg+2164c2dfg+12586bd2fg-12975cd2fg+\
     10392d3fg+3011bcefg+13302c2efg+1628bdefg+8868cdefg-13843d2efg+9139be2fg+4\
     574ce2fg+3581de2fg+13959e3fg-9184bcf2g+463c2f2g+2411bdf2g-7750cdf2g-8799d\
     2f2g-6184bef2g-7785cef2g+378def2g-918e2f2g-4051bf3g+4587cf3g+9793df3g+924\
     3ef3g-9076f4g+1779c3g2-13280bcdg2-3789c2dg2-11754bd2g2-6959cd2g2-472d3g2+\
     15938bceg2+13741c2eg2-1031bdeg2-14277cdeg2+15656d2eg2+9368be2g2-1047ce2g2\
     +5440de2g2+11656e3g2-13339bcfg2+8138c2fg2-1414bdfg2-13285cdfg2+13344d2fg2\
     -9518befg2-1353cefg2+12992defg2-374e2fg2-5513bf2g2+13046cf2g2+10536df2g2+\
     1672ef2g2-10254f3g2-15894bcg3-418c2g3+7851bdg3-9758cdg3+1014d2g3+9576beg3\
     +535ceg3+3580deg3-12979e2g3+4430bfg3+4666cfg3+3626dfg3+14215efg3+11379f2g\
     3+5968bg4+15595cg4-12250dg4-1322eg4-3384fg4+15651g5
  ↦ j[189]=ceg10-15251cde2f+7878d2e2f-10350be3f-3236ce3f-1674de3f+9235e4f+517\
     7c3f2-8412bcdf2+4614c2df2-512bd2f2+5940cd2f2-6677d3f2+4966bcef2-1019c2ef2\
     -115bdef2+13039cdef2+26d2ef2+14486be2f2-720ce2f2+12170de2f2-9812e3f2-2096\
     bcf3-3980c2f3+5151bdf3-5825cdf3+11338d2f3+15111bef3-3675cef3+10595def3+42\
     75e2f3-10822bf4-9985cf4+5615df4+12762ef4+10107f5+9915c4g-15941bd3g-3337cd\
     3g+12487d4g-9765c3eg+830bcdeg+13174c2deg+7466bd2eg-15370cd2eg+10768d3eg+7\
     625bce2g+3049c2e2g-299bde2g-11640cde2g-3239d2e2g-9131be3g-5917ce3g+8142de\
     3g-133e4g-4218c3fg+3656bcdfg-9181c2dfg-8506bd2fg-3484cd2fg+13456d3fg+1488\
     4bcefg-2221c2efg+5195bdefg-4391cdefg+15008d2efg-1949be2fg+9010ce2fg+23de2\
     fg+7651e3fg+6699bcf2g-11145c2f2g+12655bdf2g+5951cdf2g+6850d2f2g+15255bef2\
     g-10573cef2g-4316def2g-8532e2f2g-4948bf3g-13568cf3g+8601df3g-9018ef3g-787\
     0f4g-3359c3g2-5292bcdg2-15994c2dg2+9700bd2g2+11950cd2g2+3740d3g2+15451bce\
     g2+3487c2eg2+11120bdeg2-14914cdeg2-3911d2eg2+1470be2g2+10322ce2g2-8637de2\
     g2-9741e3g2+4750bcfg2-12460c2fg2-13771bdfg2+10082cdfg2+14720d2fg2+13945be\
     fg2-11324cefg2+15993defg2-15380e2fg2+1057bf2g2+11055cf2g2-14487df2g2-4618\
     ef2g2-11577f3g2-1146bcg3-13354c2g3-5124bdg3-2123cdg3-5378d2g3-6303beg3-15\
     86ceg3-15737deg3-2976e2g3+7397bfg3-6870cfg3+18dfg3-2305efg3-8327f2g3-1014\
     6bg4+15445cg4+8190dg4-4116eg4+6023fg4+9061g5
  ↦ j[190]=beg10+5105cde2f+262d2e2f+12399be3f+13634ce3f+6856de3f-10582e4f-119\
     72c3f2+5108bcdf2-4979c2df2+2986bd2f2+13656cd2f2+8114d3f2+5745bcef2-4486c2\
     ef2-9718bdef2-12151cdef2-5381d2ef2-3077be2f2-12636ce2f2+7885de2f2+7795e3f\
     2+7038bcf3+377c2f3+10026bdf3+2004cdf3+5010d2f3-6288bef3+12212cef3-6100def\
     3-8253e2f3-966bf4-5803cf4-1158df4-9499ef4+6680f5+10481c4g+3201bd3g+4177cd\
     3g+5326d4g+10591c3eg+12957bcdeg-12911c2deg-5234bd2eg-1119cd2eg-4461d3eg+7\
     622bce2g-8120c2e2g-12376bde2g-4785cde2g+14799d2e2g-7898be3g-9094ce3g-1500\
     1de3g+7642e4g+5864c3fg+13556bcdfg+13638c2dfg-7404bd2fg-6430cd2fg+271d3fg-\
     1528bcefg-4472c2efg+6675bdefg-9859cdefg-15505d2efg+3604be2fg-2858ce2fg-10\
     075de2fg+2380e3fg+13471bcf2g-4959c2f2g-7738bdf2g-9631cdf2g+3383d2f2g-4528\
     bef2g+3607cef2g-10871def2g-1276e2f2g+11009bf3g-13540cf3g+9127df3g+3979ef3\
     g-4687f4g-6869c3g2-9094bcdg2-6307c2dg2+10873bd2g2+4042cd2g2+7876d3g2+1230\
     5bceg2-8543c2eg2+2388bdeg2-4566cdeg2+6570d2eg2-10220be2g2-10923ce2g2-1135\
     7de2g2+14331e3g2+5696bcfg2+6446c2fg2+390bdfg2-12169cdfg2+15478d2fg2-6041b\
     efg2-5297cefg2+7610defg2-590e2fg2+5988bf2g2+1840cf2g2-3555df2g2+11155ef2g\
     2+14693f3g2+9157bcg3-6806c2g3+15684bdg3+13927cdg3-5031d2g3-8408beg3-14443\
     ceg3-1065deg3+15769e2g3+12518bfg3-8027cfg3-4354dfg3+4298efg3+9770f2g3+613\
     0bg4-12036cg4+5847dg4+4630eg4+2227fg4+1225g5
  ↦ j[191]=d2g10+13662cde2f+3477d2e2f-6791be3f-14774ce3f+2272de3f-13275e4f-70\
     25c3f2+3801bcdf2+7323c2df2+5260bd2f2-655cd2f2-11160d3f2-11435bcef2-15573c\
     2ef2+7035bdef2+7300cdef2+10085d2ef2+10516be2f2+10027ce2f2+1080de2f2+15560\
     e3f2+4680bcf3+1837c2f3-14104bdf3+11555cdf3-13550d2f3-7234bef3-3253cef3+13\
     114def3-8041e2f3+8961bf4+13637cf4+7859df4-10293ef4-14791f5+13881c4g+4019b\
     d3g+11936cd3g-7332d4g+1589c3eg+2694bcdeg-4215c2deg-3161bd2eg-15120cd2eg+1\
     0435d3eg-11781bce2g+1301c2e2g+287bde2g-6507cde2g-10296d2e2g+13612be3g+734\
     7ce3g+643de3g-10893e4g-12086c3fg-10880bcdfg+1494c2dfg-4554bd2fg+117cd2fg-\
     6711d3fg+6143bcefg-8964c2efg-14159bdefg-4041cdefg-11951d2efg-10877be2fg-8\
     156ce2fg+12860de2fg-9424e3fg+3557bcf2g+14640c2f2g-3202bdf2g+11999cdf2g-12\
     822d2f2g-4065bef2g+12326cef2g+9663def2g+11905e2f2g+11303bf3g+12780cf3g+27\
     3df3g+6795ef3g-8809f4g+10126c3g2+4869bcdg2-667c2dg2+6912bd2g2+7308cd2g2-5\
     122d3g2-4927bceg2+12491c2eg2+2983bdeg2-2767cdeg2+144d2eg2-12628be2g2+1454\
     7ce2g2+4063de2g2+13518e3g2+10390bcfg2-9866c2fg2+13924bdfg2+13191cdfg2+277\
     3d2fg2+10440befg2-8133cefg2+672defg2-4765e2fg2+6553bf2g2-876cf2g2+7674df2\
     g2+14226ef2g2+10597f3g2+10218bcg3-11649c2g3-15626bdg3+9857cdg3-470d2g3+17\
     79beg3+13268ceg3+611deg3-1975e2g3+1041bfg3-11546cfg3+1986dfg3-7812efg3+97\
     29f2g3-15553bg4+12071cg4+14484dg4+14198eg4-2126fg4-2804g5
  ↦ j[192]=cdg10-4785cde2f+6397d2e2f-4426be3f+11387ce3f+555de3f-13838e4f-5793\
     c3f2-2358bcdf2-8805c2df2-4134bd2f2+5817cd2f2+6958d3f2+7064bcef2-12745c2ef\
     2-3456bdef2+4439cdef2-14251d2ef2+15013be2f2-13661ce2f2+7447de2f2+3000e3f2\
     -6008bcf3+735c2f3+12975bdf3-12907cdf3-14447d2f3-10170bef3+5267cef3-6724de\
     f3-8197e2f3-13239bf4+2286cf4+5154df4+3764ef4+12645f5+58c4g+13990bd3g-992c\
     d3g+5529d4g-14746c3eg-3065bcdeg-1719c2deg+4428bd2eg-4071cd2eg-1825d3eg+40\
     66bce2g+9897c2e2g+10045bde2g-6287cde2g-11467d2e2g-5432be3g-13071ce3g+1133\
     1de3g-5388e4g-4944c3fg+12904bcdfg+4588c2dfg+6291bd2fg+3392cd2fg+8981d3fg+\
     14634bcefg-14582c2efg-5586bdefg+3692cdefg+9517d2efg-13314be2fg+4718ce2fg+\
     2869de2fg-7600e3fg-9554bcf2g+8466c2f2g+5683bdf2g-7233cdf2g-11199d2f2g-3be\
     f2g-9289cef2g+9575def2g+11607e2f2g+11916bf3g-321cf3g-3401df3g+13396ef3g+2\
     563f4g-1995c3g2+4250bcdg2+6241c2dg2-12917bd2g2-14805cd2g2-11618d3g2+8691b\
     ceg2+15654c2eg2-2978bdeg2-10645cdeg2-7504d2eg2+4498be2g2+7013ce2g2-654de2\
     g2+11169e3g2-12576bcfg2-12775c2fg2-13116bdfg2+6954cdfg2-8693d2fg2+1801bef\
     g2+10565cefg2-4671defg2-11359e2fg2-2584bf2g2-5348cf2g2-14739df2g2-3772ef2\
     g2-6822f3g2+15900bcg3-8073c2g3+878bdg3-752cdg3-15586d2g3+9665beg3-164ceg3\
     +9430deg3-4277e2g3-5238bfg3+1600cfg3-10306dfg3-4050efg3+282f2g3-2535bg4-4\
     253cg4-8968dg4+5893eg4+11460fg4+5813g5
  ↦ j[193]=bdg10+10218cde2f-11697d2e2f+9566be3f-10109ce3f-8924de3f-2809e4f+14\
     914c3f2+14972bcdf2+10295c2df2+10439bd2f2-6549cd2f2+15468d3f2+1474bcef2-16\
     69c2ef2+10434bdef2+7880cdef2-15461d2ef2-5101be2f2+7080ce2f2+1305de2f2+143\
     5e3f2-6352bcf3-142c2f3-12328bdf3+3345cdf3+3901d2f3-4354bef3+8013cef3+6330\
     def3-3379e2f3-9302bf4+1414cf4+8127df4-3040ef4+12419f5-7874c4g+7662bd3g+14\
     401cd3g+8134d4g-3432c3eg-10827bcdeg-15621c2deg+6613bd2eg+3716cd2eg+6531d3\
     eg+15950bce2g+9865c2e2g+11087bde2g+5745cde2g-1491d2e2g-1416be3g+12891ce3g\
     +14217de3g+280e4g-8468c3fg-14389bcdfg+4159c2dfg+4661bd2fg+2010cd2fg+3826d\
     3fg-9591bcefg-1807c2efg+4445bdefg-10106cdefg+1851d2efg-10225be2fg-15194ce\
     2fg-13722de2fg-5357e3fg+15718bcf2g-8395c2f2g-9993bdf2g-12222cdf2g+3303d2f\
     2g+13220bef2g-2886cef2g-11823def2g-1812e2f2g+13252bf3g-15016cf3g-3397df3g\
     +12121ef3g-5774f4g+4879c3g2-8148bcdg2+6498c2dg2-11253bd2g2-300cd2g2-7550d\
     3g2+1263bceg2-7259c2eg2+15761bdeg2-2405cdeg2+11137d2eg2+5755be2g2+11360ce\
     2g2+15753de2g2+7972e3g2-2958bcfg2-9792c2fg2+15213bdfg2-15086cdfg2+14103d2\
     fg2+9830befg2+3760cefg2-2045defg2-4759e2fg2+13776bf2g2-15334cf2g2-9002df2\
     g2+13052ef2g2-570f3g2-12644bcg3-4630c2g3+3488bdg3-818cdg3+8887d2g3+1921be\
     g3+3568ceg3+11826deg3-8170e2g3-15493bfg3-4888cfg3-6533dfg3-4166efg3+13089\
     f2g3+1879bg4-4280cg4+9296dg4+3228eg4+6497fg4-12333g5
  ↦ j[194]=c2g10-1828cde2f+13065d2e2f-2433be3f+1428ce3f-7182de3f+13221e4f-143\
     38c3f2+7163bcdf2+8380c2df2+13024bd2f2+5278cd2f2+12411d3f2-11576bcef2+1254\
     4c2ef2+3275bdef2-12125cdef2+1532d2ef2+11570be2f2+5514ce2f2-7457de2f2-3466\
     e3f2-14613bcf3-5497c2f3+3386bdf3-5116cdf3-7766d2f3+2726bef3-11262cef3-111\
     05def3+5224e2f3+2644bf4-188cf4+4857df4-452ef4+14074f5-8984c4g-9459bd3g+12\
     373cd3g-4575d4g+9127c3eg+15411bcdeg+11364c2deg+3728bd2eg-593cd2eg-8658d3e\
     g-9072bce2g-914c2e2g+2014bde2g+9742cde2g+14116d2e2g-4676be3g+13076ce3g-78\
     05de3g-6558e4g+13997c3fg-7004bcdfg-14394c2dfg+13167bd2fg+12258cd2fg-12509\
     d3fg-11030bcefg-11663c2efg-7411bdefg+1618cdefg+7763d2efg+1459be2fg-10269c\
     e2fg+5343de2fg+14428e3fg-13752bcf2g+10853c2f2g+12640bdf2g+12008cdf2g+1265\
     d2f2g-11451bef2g+644cef2g-4258def2g-10069e2f2g+9421bf3g-10949cf3g+4383df3\
     g+1668ef3g-10415f4g+14079c3g2+5366bcdg2+4540c2dg2-4442bd2g2-11837cd2g2-39\
     73d3g2-15070bceg2+11563c2eg2-613bdeg2+12298cdeg2-513d2eg2+15989be2g2-8317\
     ce2g2-4868de2g2-8161e3g2+8988bcfg2+9211c2fg2+10177bdfg2+10556cdfg2-837d2f\
     g2+11950befg2+4199cefg2+9946defg2-4333e2fg2-6720bf2g2+13458cf2g2+7083df2g\
     2-15178ef2g2+5363f3g2-13195bcg3-15250c2g3+14328bdg3+11391cdg3-9837d2g3-40\
     48beg3-12935ceg3+7798deg3-13203e2g3+10456bfg3+684cfg3+10501dfg3+14016efg3\
     -7082f2g3-314bg4+11186cg4-2201dg4+5933eg4-8111fg4+15680g5
  ↦ j[195]=bcg10-9665cde2f-15815d2e2f+8299be3f-1294ce3f-10201de3f-4562e4f-373\
     0c3f2+15049bcdf2-8055c2df2+3617bd2f2-7314cd2f2+4705d3f2+264bcef2+13849c2e\
     f2-9333bdef2-10294cdef2+6629d2ef2+4319be2f2+13411ce2f2+15279de2f2-14379e3\
     f2-13606bcf3-14150c2f3-9444bdf3-427cdf3-11430d2f3-7896bef3-2120cef3-4287d\
     ef3+5298e2f3-10291bf4+13810cf4+9792df4-2612ef4+2835f5+10279c4g-6586bd3g+3\
     205cd3g-881d4g+728c3eg-5567bcdeg+13474c2deg+4085bd2eg-8602cd2eg+3016d3eg-\
     10087bce2g-4997c2e2g-7048bde2g+11295cde2g+4828d2e2g-5055be3g+14244ce3g-17\
     97de3g+12760e4g-13878c3fg+1862bcdfg+3534c2dfg+1910bd2fg+9166cd2fg-415d3fg\
     +5970bcefg+6673c2efg+11379bdefg-15498cdefg+3788d2efg-4970be2fg-496ce2fg+1\
     859de2fg-3390e3fg+15141bcf2g-12287c2f2g-13956bdf2g-13034cdf2g-13787d2f2g+\
     5106bef2g+5979cef2g+8168def2g-10959e2f2g-7715bf3g+7501cf3g+12774df3g+8499\
     ef3g+8618f4g+14464c3g2-14505bcdg2-12733c2dg2-7255bd2g2-1179cd2g2-7218d3g2\
     +12343bceg2+12313c2eg2-6087bdeg2-5842cdeg2+10414d2eg2-1420be2g2+11463ce2g\
     2-12575de2g2-8724e3g2+2904bcfg2-7040c2fg2+15068bdfg2+11292cdfg2+11133d2fg\
     2+11228befg2-15588cefg2+4098defg2+4353e2fg2-4213bf2g2+6638cf2g2-7df2g2-13\
     252ef2g2+956f3g2-11891bcg3-15929c2g3-6092bdg3+14491cdg3-14239d2g3-9876beg\
     3+2589ceg3-5439deg3-4649e2g3+15948bfg3+11287cfg3-7048dfg3+1866efg3-1696f2\
     g3+6397bg4+14334cg4+15765dg4-2687eg4+15387fg4-10635g5
  ↦ j[196]=f3g9-4274cde2f+14326d2e2f-3580be3f-1938ce3f-15483de3f-9021e4f-731c\
     3f2+12709bcdf2+13986c2df2-1239bd2f2-11540cd2f2-14784d3f2+4685bcef2+13644c\
     2ef2+10835bdef2-6065cdef2-9376d2ef2+15904be2f2+794ce2f2+10163de2f2+14972e\
     3f2-10794bcf3+8841c2f3-392bdf3-14473cdf3-8432d2f3+10214bef3-14227cef3+368\
     2def3-14300e2f3-10658bf4+8465cf4-8009df4-15930ef4-9883f5-10087c4g+14230bd\
     3g+55cd3g+4084d4g+3132c3eg-15459bcdeg+9015c2deg-966bd2eg+15469cd2eg-2685d\
     3eg-6478bce2g+12631c2e2g+5819bde2g-2312cde2g+9984d2e2g-5466be3g-9910ce3g-\
     10175de3g-10250e4g-2825c3fg-3973bcdfg-9082c2dfg-6135bd2fg+12347cd2fg-7073\
     d3fg-10173bcefg-15551c2efg+142bdefg+309cdefg+14403d2efg-9783be2fg+8746ce2\
     fg+3300de2fg+5803e3fg+14896bcf2g-15177c2f2g-9487bdf2g+2015cdf2g+12076d2f2\
     g-1948bef2g-11802cef2g+6947def2g+8329e2f2g+3946bf3g+454cf3g+11591df3g+621\
     9ef3g-3114f4g+12813c3g2-1668bcdg2+3460c2dg2-2987bd2g2-1160cd2g2+5816d3g2+\
     3548bceg2-9936c2eg2-4622bdeg2-8963cdeg2-15946d2eg2-11981be2g2+13440ce2g2-\
     11091de2g2-8865e3g2+3832bcfg2+13835c2fg2-9855bdfg2-4366cdfg2-15989d2fg2+2\
     022befg2-14411cefg2+8971defg2+7368e2fg2+3432bf2g2+3313cf2g2-2474df2g2-933\
     3ef2g2+12052f3g2-15208bcg3+9529c2g3+9874bdg3+5385cdg3+13804d2g3-9614beg3+\
     8640ceg3-1026deg3+15121e2g3-10110bfg3-735cfg3-9533dfg3-12045efg3-8543f2g3\
     +14485bg4-14425cg4-14670dg4+114eg4-6939fg4+5509g5
  ↦ j[197]=ef2g9+5946cde2f-10535d2e2f-13420be3f+11945ce3f+11508de3f-6708e4f+5\
     793c3f2+12783bcdf2+6610c2df2-9980bd2f2+11599cd2f2-3397d3f2+6011bcef2-5532\
     c2ef2-1508bdef2-6453cdef2+11511d2ef2+10641be2f2-3939ce2f2-15137de2f2+1154\
     2e3f2-5793bcf3-2894c2f3+8624bdf3+9157cdf3+5778d2f3-13634bef3-1411cef3+789\
     9def3+15561e2f3+1989bf4-1391cf4+13679df4+1847ef4-2426f5-8876c4g-11302bd3g\
     +2989cd3g-14228d4g-6151c3eg+253bcdeg+10573c2deg+3148bd2eg-15412cd2eg+1046\
     9d3eg-5755bce2g-5421c2e2g+1085bde2g+15609cde2g-4063d2e2g-11809be3g+6217ce\
     3g+1455de3g+3089e4g-2713c3fg-11430bcdfg+10270c2dfg-4526bd2fg-9689cd2fg+11\
     788d3fg-12632bcefg+10411c2efg-8363bdefg-980cdefg+14520d2efg+3310be2fg-596\
     6ce2fg-9819de2fg-2693e3fg-6854bcf2g+6170c2f2g+2499bdf2g-3296cdf2g+10174d2\
     f2g-13654bef2g+11281cef2g-5325def2g-11828e2f2g+4260bf3g+13271cf3g+8501df3\
     g+15048ef3g-12950f4g-1657c3g2-8332bcdg2-14567c2dg2-13684bd2g2-13190cd2g2+\
     10313d3g2-13098bceg2+12846c2eg2-2463bdeg2+11719cdeg2+9559d2eg2-4935be2g2+\
     14566ce2g2-11322de2g2-13268e3g2+4223bcfg2-1390c2fg2-2026bdfg2+3527cdfg2-8\
     830d2fg2-5133befg2-6667cefg2-11179defg2-8397e2fg2-8423bf2g2+2332cf2g2-137\
     14df2g2+13625ef2g2-12897f3g2+10678bcg3+5577c2g3+359bdg3-7624cdg3-7355d2g3\
     +13545beg3+12140ceg3+136deg3-15499e2g3-12646bfg3+14928cfg3+1654dfg3-6938e\
     fg3+6758f2g3-10020bg4+14810cg4+10878dg4+189eg4-1484fg4-6548g5
  ↦ j[198]=df2g9-14311cde2f-2723d2e2f+15071be3f-1204ce3f-4714de3f-11134e4f+15\
     681c3f2+13341bcdf2+12106c2df2-1422bd2f2-14653cd2f2-1185d3f2+8427bcef2-278\
     1c2ef2+118bdef2-15436cdef2+6436d2ef2+7153be2f2+3365ce2f2+4762de2f2+1569e3\
     f2+4618bcf3+10744c2f3-11740bdf3+2479cdf3+8109d2f3+5823bef3+9366cef3+13798\
     def3-7231e2f3-15886bf4+260cf4-6564df4-8597ef4-15909f5+3071c4g-4835bd3g-11\
     868cd3g+1330d4g+2760c3eg+14410bcdeg-15256c2deg+4631bd2eg-2093cd2eg-11076d\
     3eg-14035bce2g+2311c2e2g-13477bde2g-15062cde2g+7719d2e2g-15674be3g+8466ce\
     3g+13169de3g-1705e4g+3963c3fg-11236bcdfg-15882c2dfg-3910bd2fg-3748cd2fg-7\
     238d3fg+13173bcefg-14908c2efg+15939bdefg+11389cdefg-2841d2efg+7948be2fg+1\
     2405ce2fg+7889de2fg+7288e3fg+12451bcf2g-1908c2f2g-11704bdf2g+13377cdf2g+7\
     335d2f2g+1677bef2g+2072cef2g-10790def2g+14914e2f2g+11762bf3g+1819cf3g+145\
     59df3g+14173ef3g-14818f4g+7480c3g2-15009bcdg2+9046c2dg2-2293bd2g2+9886cd2\
     g2-12009d3g2-4260bceg2+7936c2eg2+323bdeg2+3456cdeg2+3945d2eg2-5186be2g2+7\
     06ce2g2-6940de2g2+289e3g2-9070bcfg2-9168c2fg2+6124bdfg2+1220cdfg2-6863d2f\
     g2+213befg2-15724cefg2-8541defg2-7939e2fg2-12324bf2g2-12064cf2g2+4933df2g\
     2-6002ef2g2+14653f3g2-11889bcg3+12791c2g3-9953bdg3-3002cdg3-6063d2g3-776b\
     eg3+11250ceg3-1052deg3+15661e2g3-13060bfg3-729cfg3+4535dfg3-8709efg3-1431\
     f2g3+12534bg4+12597cg4-11077dg4-5972eg4+12104fg4-12253g5
  ↦ j[199]=cf2g9+1488cde2f+1729d2e2f-6887be3f-7532ce3f+3674de3f-9295e4f-8044c\
     3f2+15597bcdf2-1424c2df2+13803bd2f2-9431cd2f2+15293d3f2+1281bcef2-9817c2e\
     f2+15299bdef2-5933cdef2-12342d2ef2+13898be2f2-14314ce2f2-4679de2f2+10572e\
     3f2+13492bcf3-13704c2f3-10422bdf3+2023cdf3-14161d2f3+9951bef3-14156cef3-7\
     422def3-67e2f3-10945bf4+489cf4+1340df4+995ef4-1729f5-8492c4g-13435bd3g+15\
     844cd3g-9915d4g-1461c3eg-11602bcdeg+13723c2deg+3052bd2eg+2532cd2eg+14289d\
     3eg+12423bce2g+3659c2e2g-9277bde2g+4218cde2g+12343d2e2g+8291be3g+13530ce3\
     g+15829de3g-4604e4g-8001c3fg+13527bcdfg-13805c2dfg+11139bd2fg+6379cd2fg+7\
     676d3fg+9947bcefg+1442c2efg+12924bdefg-7652cdefg-6524d2efg+4388be2fg-3220\
     ce2fg-791de2fg-8128e3fg+12329bcf2g-8103c2f2g+14108bdf2g-2648cdf2g+6703d2f\
     2g+6958bef2g+2931cef2g+12805def2g+12573e2f2g-7282bf3g-3922cf3g+15624df3g+\
     4057ef3g+2651f4g+5061c3g2+2487bcdg2-8683c2dg2-4142bd2g2+8439cd2g2+7768d3g\
     2-6589bceg2+321c2eg2+13527bdeg2+10958cdeg2-12821d2eg2-6643be2g2+9502ce2g2\
     +4367de2g2+12069e3g2+14315bcfg2-1810c2fg2-7807bdfg2+14470cdfg2-5025d2fg2-\
     11527befg2+13916cefg2-6209defg2-3641e2fg2+6934bf2g2+8751cf2g2+7313df2g2+1\
     1721ef2g2-12017f3g2+13911bcg3-6645c2g3+2418bdg3+9487cdg3-6957d2g3+5100beg\
     3-7555ceg3+11481deg3+13837e2g3+2489bfg3-4435cfg3+15322dfg3-6198efg3-5253f\
     2g3+12985bg4-13312cg4-11666dg4+8206eg4+4007fg4+14141g5
  ↦ j[200]=bf2g9+13425cde2f+2039d2e2f+11489be3f-10215ce3f+14770de3f-3236e4f-1\
     1008c3f2+1143bcdf2-85c2df2-11458bd2f2-2320cd2f2+7835d3f2-6606bcef2+10258c\
     2ef2-10787bdef2-2384cdef2+12522d2ef2+8217be2f2-4562ce2f2+8867de2f2-3891e3\
     f2-12794bcf3-15422c2f3-11883bdf3-7865cdf3-249d2f3+14580bef3+14874cef3-918\
     7def3-6672e2f3-6755bf4+5021cf4+3172df4+5720ef4-11532f5+4782c4g-10998bd3g-\
     3888cd3g+4811d4g+15754c3eg-6862bcdeg-9350c2deg+846bd2eg-8004cd2eg+4098d3e\
     g+2984bce2g+3654c2e2g-13166bde2g+1853cde2g+12960d2e2g-7636be3g+2596ce3g+6\
     221de3g+3589e4g-6887c3fg+9271bcdfg+9938c2dfg+6176bd2fg-13284cd2fg-14541d3\
     fg+2313bcefg+14469c2efg+8636bdefg+4944cdefg+2310d2efg-13118be2fg-9631ce2f\
     g-2246de2fg+1097e3fg-13961bcf2g-8380c2f2g-2815bdf2g+14908cdf2g+11236d2f2g\
     -7582bef2g-2950cef2g+156def2g+8724e2f2g-7984bf3g+1675cf3g+14370df3g+9060e\
     f3g+13451f4g+4827c3g2-15397bcdg2+15454c2dg2+8249bd2g2+12749cd2g2+14910d3g\
     2-3129bceg2+8987c2eg2-8994bdeg2+8185cdeg2-11847d2eg2-7159be2g2-2462ce2g2+\
     5304de2g2-14522e3g2+14179bcfg2+12947c2fg2-11963bdfg2+15631cdfg2-11018d2fg\
     2-3862befg2-14051cefg2+5610defg2-15449e2fg2+1825bf2g2-964cf2g2+1817df2g2+\
     14695ef2g2+7223f3g2-15033bcg3-11243c2g3-2329bdg3-5742cdg3-400d2g3-13488be\
     g3-4832ceg3+10896deg3+4314e2g3+11467bfg3+4414cfg3+298dfg3-2016efg3+8087f2\
     g3-3629bg4+6309cg4-11216dg4-2480eg4-9479fg4-14294g5
  ↦ j[201]=e2fg9+4533cde2f+4024d2e2f+5390be3f+15076ce3f+12634de3f-7483e4f-124\
     77c3f2+5415bcdf2-6118c2df2-15949bd2f2+13511cd2f2-12804d3f2-7737bcef2-1308\
     2c2ef2-1395bdef2+8287cdef2+12550d2ef2+9292be2f2-6492ce2f2-12616de2f2-1456\
    8e3f2-5476bcf3+15080c2f3-7856bdf3+9008cdf3-14444d2f3+5733bef3+11996cef3-8\
    07def3-3818e2f3+6227bf4+3399cf4+14162df4-2428ef4-14312f5-13036c4g-15597bd\
    3g+7222cd3g+23d4g-9858c3eg-15285bcdeg+4795c2deg+6389bd2eg+1941cd2eg+5418d\
    3eg+8904bce2g+261c2e2g+12739bde2g-15299cde2g-5800d2e2g-6589be3g-13828ce3g\
    +12911de3g-10067e4g+987c3fg+3996bcdfg-1822c2dfg-8351bd2fg+14098cd2fg+1395\
    4d3fg+11891bcefg-8123c2efg+6801bdefg-4882cdefg+1664d2efg-8054be2fg+4799ce\
    2fg-8528de2fg-10451e3fg-10433bcf2g+2511c2f2g+12486bdf2g-4974cdf2g+10783d2\
    f2g-11378bef2g+3325cef2g+11403def2g+15496e2f2g-11928bf3g+9635cf3g-11887df\
    3g+15005ef3g-12341f4g-13310c3g2-6542bcdg2-6959c2dg2-8896bd2g2+15286cd2g2-\
    5216d3g2+2317bceg2+8522c2eg2-13173bdeg2-11565cdeg2-3747d2eg2-3397be2g2-13\
    589ce2g2+7796de2g2-12156e3g2+10892bcfg2+9284c2fg2+2665bdfg2-15414cdfg2-12\
    82d2fg2+11459befg2-13444cefg2-7512defg2+15274e2fg2+1346bf2g2+5262cf2g2-93\
    df2g2-5907ef2g2-2263f3g2-69bcg3-12400c2g3+12814bdg3-3883cdg3-15506d2g3-12\
    172beg3+14983ceg3+1006deg3-11224e2g3+11259bfg3-7451cfg3-5018dfg3+8560efg3\
    +6468f2g3-9741bg4-947cg4+6362dg4-3839eg4-14738fg4-15496g5
  ↦ j[202]=defg9+8017cde2f+1227d2e2f-4585be3f-10981ce3f-12962de3f+210e4f+4437\
    c3f2+14598bcdf2-3455c2df2+3256bd2f2-3763cd2f2-1262d3f2+8772bcef2+15121c2e\
    f2-2284bdef2+1278cdef2-10164d2ef2+12927be2f2+2963ce2f2+10467de2f2+15285e3\
    f2+3698bcf3-14298c2f3+6410bdf3+10039cdf3-7081d2f3-7885bef3+3865cef3+12285\
    def3+1863e2f3-2096bf4+4563cf4+14205df4+119ef4-4865f5+13349c4g+392bd3g-352\
    1cd3g+6535d4g-10722c3eg-9416bcdeg-13791c2deg+11395bd2eg+9774cd2eg+9335d3e\
    g+4906bce2g+13021c2e2g+4116bde2g-15420cde2g-2611d2e2g+6987be3g-6600ce3g-1\
    0051de3g-6635e4g-6326c3fg+6158bcdfg+8399c2dfg-12255bd2fg+13615cd2fg+14901\
    d3fg-7796bcefg+2122c2efg+84bdefg-7563cdefg-15700d2efg-982be2fg+13763ce2fg\
    +1368de2fg-683e3fg+8607bcf2g-14421c2f2g-12447bdf2g+3087cdf2g+9841d2f2g+65\
    63bef2g-7579cef2g+15933def2g+2623e2f2g+13650bf3g+14442cf3g-13818df3g-2255\
    ef3g-5080f4g-4741c3g2-9965bcdg2-4691c2dg2+4314bd2g2+3761cd2g2+7040d3g2+11\
    294bceg2+9416c2eg2-6756bdeg2-8114cdeg2-9766d2eg2+6419be2g2-7084ce2g2-1382\
    de2g2-11221e3g2+4926bcfg2+2659c2fg2+13307bdfg2+5372cdfg2+6795d2fg2-4110be\
    fg2-10004cefg2+1449defg2-10488e2fg2+12590bf2g2-4960cf2g2+8600df2g2-15303e\
    f2g2-4816f3g2-7289bcg3+12053c2g3-1459bdg3+10100cdg3-13581d2g3-12961beg3+6\
    715ceg3-15997deg3-9684e2g3-6724bfg3-5934cfg3-13232dfg3-12869efg3-14549f2g\
    3+7647bg4+11751cg4-9936dg4-15849eg4-5781fg4+13819g5
  ↦ j[203]=cefg9+7976cde2f+10526d2e2f+3069be3f-2880ce3f+2545de3f-8262e4f+1294\
    4c3f2-2475bcdf2+5528c2df2+8956bd2f2-309cd2f2+3422d3f2-1902bcef2-8856c2ef2\
    +520bdef2-13055cdef2+4233d2ef2+9821be2f2-11742ce2f2+2146de2f2+273e3f2+117\
    49bcf3+4873c2f3-9653bdf3-4307cdf3-9979d2f3-9779bef3+9578cef3-8663def3-130\
    37e2f3+10457bf4-7373cf4-2008df4+6757ef4-11362f5+7836c4g-14571bd3g-3807cd3\
    g-13796d4g+14058c3eg+8195bcdeg+8470c2deg-13191bd2eg+431cd2eg-9066d3eg+108\
    77bce2g-9730c2e2g+1918bde2g-997cde2g+2810d2e2g+1845be3g-14321ce3g-2984de3\
    g-8844e4g-7396c3fg+15772bcdfg-14026c2dfg-14008bd2fg+2578cd2fg-1247d3fg+13\
    754bcefg-4482c2efg-11246bdefg-8416cdefg-6780d2efg+10284be2fg+1897ce2fg+42\
    9de2fg+9711e3fg-13702bcf2g+9679c2f2g-1977bdf2g+10122cdf2g-12013d2f2g+1580\
    6bef2g-1677cef2g+15225def2g+6265e2f2g+9837bf3g+3986cf3g+3143df3g+10937ef3\
    g+1674f4g-3806c3g2-2660bcdg2+8811c2dg2+8013bd2g2-14777cd2g2+10073d3g2-119\
    23bceg2-3551c2eg2+8491bdeg2-14388cdeg2-7220d2eg2+125be2g2-11144ce2g2-1412\
    de2g2+4422e3g2-3831bcfg2-5905c2fg2-271bdfg2+6247cdfg2+5412d2fg2-15308befg\
    2+13683cefg2-3519defg2-12852e2fg2-5174bf2g2+11364cf2g2-2576df2g2-7913ef2g\
    2+4153f3g2+1278bcg3+4807c2g3-14570bdg3+13135cdg3-4388d2g3+4952beg3+3597ce\
    g3+5587deg3+8179e2g3+13058bfg3-3430cfg3+2294dfg3+11914efg3+11275f2g3+9467\
    bg4-9122cg4-12164dg4-21eg4+1213fg4+1381g5
  ↦ j[204]=befg9+8727cde2f-4487d2e2f-14306be3f+11933ce3f+9868de3f-190e4f-4753\
    c3f2-6401bcdf2-1731c2df2+11675bd2f2+3081cd2f2-9578d3f2-10941bcef2+4312c2e\
    f2+3804bdef2+2661cdef2+13220d2ef2-6961be2f2+2298ce2f2-7897de2f2+10010e3f2\
    +3588bcf3+3282c2f3-10847bdf3+3458cdf3+14309d2f3+130bef3-12128cef3-12211de\
    f3+6853e2f3+7827bf4-13639cf4+12450df4-9449ef4+4690f5-9254c4g-5564bd3g-128\
    11cd3g+9706d4g-13987c3eg-8569bcdeg+9608c2deg-215bd2eg+7756cd2eg+15419d3eg\
    +11419bce2g+1951c2e2g+4021bde2g+8497cde2g-11403d2e2g+5869be3g-7948ce3g-82\
    07de3g-11028e4g-36c3fg-4810bcdfg+258c2dfg+5175bd2fg+13615cd2fg+10224d3fg+\
    972bcefg+12466c2efg-6456bdefg+4686cdefg-1335d2efg+9171be2fg+7826ce2fg+141\
    86de2fg+6221e3fg+8011bcf2g+11433c2f2g-14131bdf2g+12093cdf2g-13451d2f2g-15\
    099bef2g+2431cef2g-2344def2g-11526e2f2g+7505bf3g-13041cf3g+10848df3g-322e\
    f3g-8770f4g-822c3g2+14390bcdg2-9172c2dg2+2446bd2g2-8046cd2g2-13412d3g2-80\
    02bceg2-7167c2eg2+6371bdeg2+13426cdeg2-450d2eg2+1950be2g2-4585ce2g2+2702d\
    e2g2-1240e3g2-8005bcfg2+13560c2fg2+6530bdfg2-6078cdfg2+10709d2fg2+8200bef\
    g2-14051cefg2+8664defg2+3895e2fg2-6690bf2g2-8940cf2g2+3401df2g2+15596ef2g\
    2-5598f3g2-7171bcg3-11637c2g3-4667bdg3-1730cdg3+5518d2g3-14885beg3-3730ce\
    g3+11875deg3-11682e2g3+7697bfg3-7111cfg3-8705dfg3+12827efg3+12979f2g3-503\
    5bg4-15135cg4+10615dg4-15807eg4+3410fg4-10926g5
 ↦ j[205]=d2fg9-6222cde2f+14810d2e2f+2100be3f+13557ce3f+7815de3f+477e4f-9041\
    c3f2+8961bcdf2+1800c2df2-13633bd2f2-13923cd2f2+13677d3f2-2947bcef2-3923c2\
    ef2-15829bdef2-2599cdef2+5859d2ef2+10258be2f2-10999ce2f2-12269de2f2+10885\
    e3f2+15411bcf3+6856c2f3+5370bdf3-15937cdf3+12990d2f3+9578bef3-4974cef3-15\
    717def3-11247e2f3-4991bf4-7094cf4-13446df4-9158ef4+11807f5+3386c4g-4539bd\
    3g-1637cd3g-6968d4g-12403c3eg+197bcdeg-15443c2deg-6207bd2eg-13380cd2eg-53\
    76d3eg-7424bce2g-13998c2e2g-11622bde2g+1382cde2g+7904d2e2g+15809be3g-4536\
    ce3g+6248de3g+6471e4g+8663c3fg+13581bcdfg+982c2dfg-5603bd2fg-8811cd2fg-15\
    263d3fg-10109bcefg-10677c2efg-12941bdefg-12735cdefg-7290d2efg+16be2fg+969\
    5ce2fg-7211de2fg+7622e3fg-8143bcf2g+149c2f2g+7444bdf2g-1826cdf2g+8041d2f2\
    g+15876bef2g-1725cef2g-6677def2g+4961e2f2g-14394bf3g+4271cf3g+9066df3g+51\
    5ef3g-13992f4g-6783c3g2-1126bcdg2+1369c2dg2+9456bd2g2-7750cd2g2+10740d3g2\
    -11260bceg2-8096c2eg2-4822bdeg2-10020cdeg2-13218d2eg2-12175be2g2-6565ce2g\
    2-15151de2g2+2257e3g2-6599bcfg2-5767c2fg2-9902bdfg2+14154cdfg2+13139d2fg2\
    +7451befg2+8476cefg2-7746defg2-14273e2fg2+14982bf2g2-9766cf2g2+8962df2g2-\
    4539ef2g2-2870f3g2-1066bcg3-13743c2g3-14291bdg3+1601cdg3-5414d2g3+10118be\
    g3+3039ceg3-15108deg3+14978e2g3+4005bfg3+6074cfg3-2366dfg3+9459efg3+966f2\
    g3-2993bg4+5185cg4+9380dg4+9902eg4+14953fg4+3136g5
 ↦ j[206]=cdfg9+5307cde2f+3835d2e2f-6923be3f+4482ce3f+13400de3f+15227e4f-123\
    29c3f2+12006bcdf2+1219c2df2+3998bd2f2+9417cd2f2+4083d3f2+13550bcef2-9939c\
    2ef2-7775bdef2-9423cdef2-4631d2ef2-846be2f2-3523ce2f2-1513de2f2+14177e3f2\
    +3102bcf3-4252c2f3-6413bdf3-13057cdf3+2d2f3-12870bef3-1495cef3+2656def3+1\
    5370e2f3-11923bf4+15003cf4-4909df4+13962ef4-1109f5+8374c4g+7766bd3g+4850c\
    d3g-9254d4g+4152c3eg+10129bcdeg-9227c2deg-13433bd2eg-5817cd2eg-463d3eg+12\
    567bce2g-4801c2e2g-15957bde2g-10131cde2g+6899d2e2g+7098be3g-8784ce3g+726d\
    e3g-3874e4g-5915c3fg+2789bcdfg-6357c2dfg+10842bd2fg-2932cd2fg-3800d3fg+77\
    30bcefg+7291c2efg+7272bdefg-2077cdefg+2927d2efg+13341be2fg-14888ce2fg+666\
    6de2fg+7723e3fg+11240bcf2g-14026c2f2g-5869bdf2g+15924cdf2g+13255d2f2g-383\
    7bef2g-9526cef2g+3994def2g+4825e2f2g-1803bf3g-11467cf3g-14271df3g+10069ef\
    3g+5483f4g+4703c3g2-1764bcdg2+4439c2dg2-10287bd2g2-4969cd2g2-12947d3g2+79\
    87bceg2+14828c2eg2-15790bdeg2+14546cdeg2-377d2eg2-5842be2g2-10113ce2g2-56\
    21de2g2+1236e3g2+11439bcfg2-15804c2fg2-7669bdfg2-4255cdfg2+861d2fg2-8287b\
    efg2+5627cefg2-1530defg2-4071e2fg2-8372bf2g2-9737cf2g2-5635df2g2-12054ef2\
    g2-14407f3g2-6702bcg3+11675c2g3+4584bdg3-12688cdg3-9704d2g3+13923beg3-649\
    2ceg3-7473deg3+14348e2g3+11313bfg3-7987cfg3+13638dfg3-10211efg3-7052f2g3+\
    11689bg4-2392cg4-9899dg4-57eg4+2221fg4+5838g5
 ↦ j[207]=bdfg9+6180cde2f-9339d2e2f+8979be3f+6372ce3f+11737de3f-9558e4f+9770\
    c3f2-14241bcdf2+9719c2df2-10795bd2f2+4475cd2f2-2065d3f2-13082bcef2-8285c2\
    ef2-10736bdef2-2786cdef2+15276d2ef2+15607be2f2+8731ce2f2+14527de2f2+14128\
    e3f2+15731bcf3-5355c2f3-5162bdf3+7394cdf3+5457d2f3+4428bef3-3101cef3+1049\
    5def3-13915e2f3+14081bf4+7734cf4+7484df4-10172ef4+15701f5-11628c4g+13103b\
    d3g-9967cd3g+9947d4g+11132c3eg+10800bcdeg+14426c2deg+14031bd2eg-7858cd2eg\
    +15643d3eg-13026bce2g+7955c2e2g-15976bde2g-13251cde2g+14246d2e2g-12742be3\
    g-195ce3g-1809de3g+3685e4g-14201c3fg+8075bcdfg+8880c2dfg-1398bd2fg+6422cd\
    2fg+2209d3fg+14979bcefg+3213c2efg-1979bdefg-7917cdefg-13585d2efg+7580be2f\
    g-3586ce2fg-6563de2fg-1486e3fg+7993bcf2g-6209c2f2g+4203bdf2g+2842cdf2g-15\
    08d2f2g-10549bef2g+5932cef2g+13568def2g-4223e2f2g+8614bf3g+13922cf3g-5675\
    df3g+4569ef3g-4178f4g+8142c3g2-4610bcdg2+13667c2dg2+12825bd2g2-1837cd2g2+\
    2456d3g2-10934bceg2+3282c2eg2+10757bdeg2-13642cdeg2+13969d2eg2-11313be2g2\
    -2473ce2g2-4878de2g2-5566e3g2-9826bcfg2+12786c2fg2-8810bdfg2-4362cdfg2-14\
    111d2fg2+1661befg2+10742cefg2-6043defg2-9693e2fg2+444bf2g2-2535cf2g2-1576\
    df2g2+6211ef2g2-563f3g2-2601bcg3+3090c2g3+9339bdg3+11184cdg3-14444d2g3+53\
    98beg3+1504ceg3-7236deg3-10467e2g3-2032bfg3-9353cfg3-10391dfg3+3118efg3+1\
    4061f2g3+3186bg4+696cg4-2408dg4-8032eg4+265fg4+5090g5
 ↦ j[208]=c2fg9+15357cde2f+1738d2e2f-9507be3f-3626ce3f-15619de3f-686e4f-5065\
    c3f2+3822bcdf2+13158c2df2+7413bd2f2+12580cd2f2+186d3f2+12235bcef2-4050c2e\
    f2-15720bdef2+3553cdef2+1296d2ef2+8362be2f2-5718ce2f2-6287de2f2+6564e3f2-\
    55bcf3-2595c2f3+7253bdf3-4348cdf3+12975d2f3-9576bef3+9338cef3-439def3+630\
    1e2f3-4580bf4-7757cf4-11049df4-14171ef4-11055f5+6823c4g+15182bd3g+14671cd\
    3g-1971d4g-11038c3eg-4026bcdeg-3934c2deg+5706bd2eg+6924cd2eg-7699d3eg+662\
    1bce2g-8422c2e2g+4137bde2g+138cde2g-5421d2e2g-13835be3g+4635ce3g+7017de3g\
    -12335e4g+698c3fg+4667bcdfg+6565c2dfg-15372bd2fg-5252cd2fg-5550d3fg+2420b\
    cefg-10126c2efg+8356bdefg+13108cdefg+14331d2efg+8164be2fg-9618ce2fg+10187\
    de2fg+14064e3fg-6360bcf2g+8261c2f2g-15653bdf2g+2500cdf2g+15286d2f2g+12630\
    bef2g+1134cef2g-1485def2g+13809e2f2g+10758bf3g-2941cf3g-6239df3g+2134ef3g\
    -8667f4g-12788c3g2-3400bcdg2-12361c2dg2+12620bd2g2-6308cd2g2-3362d3g2+394\
    6bceg2-9285c2eg2+15716bdeg2+9124cdeg2+402d2eg2+7640be2g2-8856ce2g2-4035de\
    2g2+12314e3g2+8662bcfg2-6c2fg2+14899bdfg2+8970cdfg2-15927d2fg2-4785befg2-\
    7609cefg2-4623defg2-12648e2fg2+10696bf2g2+7440cf2g2-5896df2g2-1020ef2g2-2\
    383f3g2+9680bcg3+3792c2g3+14326bdg3+2318cdg3+3776d2g3+1059beg3+1827ceg3-1\
    0932deg3-704e2g3-11362bfg3+4423cfg3+2162dfg3-2768efg3+15279f2g3-6730bg4-8\
    072cg4-9216dg4-1689eg4-5244fg4+9733g5
 ↦ j[209]=bcfg9-15466cde2f+6718d2e2f+2038be3f+10674ce3f+13682de3f-6829e4f+83\
    43c3f2-4541bcdf2+9354c2df2+8282bd2f2-3178cd2f2-7003d3f2+3456bcef2-7375c2e\
    f2+9430bdef2-3091cdef2+7470d2ef2+4264be2f2-19ce2f2-9291de2f2-6891e3f2+599\
    3bcf3-11666c2f3-10102bdf3-4073cdf3+4000d2f3+12538bef3-12510cef3+14623def3\
    -8160e2f3+14488bf4-13019cf4-5561df4+2689ef4-11865f5-11270c4g+15426bd3g+80\
    9cd3g+8769d4g-7015c3eg+4100bcdeg+14874c2deg-8357bd2eg-5262cd2eg+8776d3eg-\
    5117bce2g-5492c2e2g-15245bde2g-2917cde2g+1659d2e2g-15105be3g+13591ce3g-62\
    63de3g+4536e4g-12534c3fg+10482bcdfg+14576c2dfg+13581bd2fg-13178cd2fg-8603\
    d3fg+10446bcefg-7576c2efg-12276bdefg+681cdefg+9729d2efg+1442be2fg-3453ce2\
    fg+15537de2fg+9104e3fg+687bcf2g+7922c2f2g+1650bdf2g-15043cdf2g-8009d2f2g-\
    13581bef2g-11867cef2g+6777def2g-9064e2f2g-12354bf3g+9777cf3g-6641df3g-156\
    46ef3g-9695f4g-13928c3g2+4079bcdg2+2185c2dg2-6802bd2g2+2989cd2g2-14799d3g\
    2+12629bceg2-9283c2eg2-1210bdeg2-12524cdeg2+10425d2eg2+10233be2g2+15276ce\
    2g2+14325de2g2-10439e3g2+15071bcfg2+12031c2fg2-9476bdfg2+14032cdfg2+13145\
    d2fg2+14274befg2-13820cefg2-7161defg2+8914e2fg2-5803bf2g2+10672cf2g2-1302\
    7df2g2+15161ef2g2-9575f3g2+9221bcg3+11806c2g3+13000bdg3+13105cdg3-9239d2g\
    3+13401beg3+14070ceg3+11780deg3-11901e2g3-6136bfg3+4976cfg3-6049dfg3+1281\
    0efg3+13364f2g3+11902bg4-7739cg4+5263dg4+9970eg4+2637fg4+1403g5
```

```
↦ j[210]=e3g9+15555cde2f+11769d2e2f+4308be3f+6313ce3f+11996de3f+13935e4f+11\
   623c3f2-587bcdf2-4916c2df2+9206bd2f2-10060cd2f2-13013d3f2-4804bcef2+8053c\
   2ef2+8661bdef2-13535cdef2-9527d2ef2+11151be2f2+10594ce2f2-11871de2f2+1763\
   e3f2+1685bcf3-15298c2f3-2831bdf3+14261cdf3-8241d2f3+11672bef3-1507cef3+10\
   427def3-764e2f3-1787bf4+13086cf4+14455df4-14040ef4+7416f5+11291c4g+7242bd\
   3g-8624cd3g-2548d4g-10683c3eg-2397bcdeg+14264c2deg+5549bd2eg-12554cd2eg-3\
   375d3eg-8187bce2g+8266c2e2g-3850bde2g-13489cde2g-8750d2e2g-8147be3g-8152c\
   e3g-6902de3g+8888e4g-15074c3fg-10085bcdfg+2363c2dfg+11553bd2fg+11292cd2fg\
   -12202d3fg-14773bcefg+2352c2efg-2828bdefg+7381cdefg-7196d2efg+6418be2fg+9\
   596ce2fg-2265de2fg-5030e3fg-10171bcf2g+8869c2f2g-11409bdf2g+7650cdf2g-485\
   4d2f2g-12309bef2g+3367cef2g+2721def2g+7867e2f2g+14363bf3g+8216cf3g+670df3\
   g-12060ef3g+13376f4g+1412c3g2-10356bcdg2+807c2dg2-10724bd2g2-4786cd2g2-51\
   35d3g2+6883bceg2-14728c2eg2+3072bdeg2-8507cdeg2+553d2eg2+13400be2g2+528ce\
   2g2-5642de2g2-3591e3g2+4388bcfg2-1317c2fg2+11566bdfg2-4950cdfg2+11939d2fg\
   2+13761befg2+15113cefg2+14648defg2-13028e2fg2-8178bf2g2+13704cf2g2+15789d\
   f2g2+6129ef2g2+8677f3g2-14166bcg3+4040c2g3-3326bdg3+13419cdg3-14011d2g3-7\
   341beg3-10187ceg3-14478deg3-7260e2g3-4393bfg3+875cfg3-12787dfg3-7295efg3+\
   10106f2g3+7771bg4+11383cg4-10479dg4+1394eg4+8625fg4-12103g5
```

See also: Section D.4.31.4 [modJanet], page 1375.

D.4.31.4 modJanet

Procedure from library `rstandard.lib` (see Section D.4.31 [rstandard_lib], page 1265).

Usage: modJanet(I,i); I is an ideal, i an integer (optional).

Return: ideal, a Janet basis for I using modular methods.

Purpose: Computes a Janet basis for the ideal given by the generators in I using modular techniques.
If second argument is 0 then the result is not verified.

Example:

```
LIB "rstandard.lib";
ring R=0,(t,x,y,z),ds;
ideal i=
5t3x2z+2t2y3x5,
7y+4x2y+y2x+2zt,
3tz+3yz2+2yz4;
ideal j=modJanet(i);  j;
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ Length of Janet basis: 3
↦ j[1]=y
↦ j[2]=tz
↦ j[3]=ty
ring S=0,(x,y,z),dp;
poly p1 =x2y*(47x5y7z3+28xy5z8+63+91x5y3z7);
poly p2 =xyz*(57y6+21x2yz9+51y2z2+15x2z4);
poly p3 =xy4z*(74y+32x6z7+53x5y2z+17x2y3z);
poly p4 =y3z*(21x2z6+32x10y6z5+23x5y5z7+27y2);
poly p5 =xz*(36y2z2+81x9y10+19x2y5z4+79x4z6);
ideal i =p1,p2,p3,p4,p5;
ideal j=modJanet(i,0);  j;
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ Length of Janet basis: 9
↦ j[1]=x2y
↦ j[2]=x3y
↦ j[3]=x4y
↦ j[4]=y5z
↦ j[5]=x5y
↦ j[6]=xy3z3
↦ j[7]=xy5z
↦ j[8]=xy4z3
↦ j[9]=x5z7+36/79xy2z3
```

See also: Section D.4.31.3 [rJanet], page 1305.

D.4.32 sagbi_lib

Library: sagbi.lib

Purpose: Compute SAGBI basis (subalgebra bases analogous to Groebner bases for ideals) of a subalgebra

Authors: Jan Hackfeld, Jan.Hackfeld@rwth-aachen.de
Gerhard Pfister, pfister@mathematik.uni-kl.de
Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview: SAGBI stands for 'subalgebra bases analogous to Groebner bases for ideals'. SAGBI bases provide important tools for working with finitely presented subalgebras of a polynomial ring. Note, that in contrast to Groebner bases, SAGBI bases may be infinite.

References:

Ana Bravo: Some Facts About Canonical Subalgebra Bases, MSRI Publications 51, p. 247-254

Procedures: See also: Section D.4.2 [algebra_lib], page 1003.

D.4.32.1 sagbiSPoly

Procedure from library `sagbi.lib` (see Section D.4.32 [sagbi_lib], page 1376).

Usage: sagbiSPoly(A[, returnRing, meth]); A is an ideal, returnRing and meth are integers.

Return: ideal or ring

Assume: basering is not a qring

Purpose: Returns SAGBI S-polynomials of the leading terms of a given ideal A if returnRing=0.
Otherwise returns a new ring containing the ideals algebraicRelations
and spolynomials, where these objects are explained by their name.
See the example on how to access these objects.
The other optional argument meth determines which method is
used for computing the algebraic relations.
- If meth=0 (default), the procedure std is used.
- If meth=1, the procedure slimgb is used.
- If meth=2, the procedure uses toric_ideal.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y),dp;
ideal A=x*y+x,x*y^2,y^2+y,x^2+x;
//------------------ Compute the SAGBI S-polynomials only
sagbiSPoly(A);
↦ _[1]=x2y-xy2+x2-xy
↦ _[2]=x2y3+1/2xy4+1/2x2y2+xy3+1/2xy2
//------------------ Extended ring is to be returned, which contains
// the ideal of algebraic relations and the ideal of the S-polynomials
def rNew=sagbiSPoly(A,1);  setring rNew;
spolynomials;
↦ spolynomials[1]=x^2*y-x*y^2+x^2-x*y
↦ spolynomials[2]=x^2*y^3+1/2*x*y^4+1/2*x^2*y^2+x*y^3+1/2*x*y^2
algebraicRelations;
↦ algebraicRelations[1]=@y(1)^2-@y(3)*@y(4)
↦ algebraicRelations[2]=@y(3)^2*@y(4)-@y(2)^2
//----------------- Now we verify that the substitution of A[i] into @y(i)
// results in the spolynomials listed above
ideal A=fetch(r,A);
map phi=rNew,x,y,A;
ideal spolynomials2=simplify(phi(algebraicRelations),1);
spolynomials2;
↦ spolynomials2[1]=x^2*y-x*y^2+x^2-x*y
↦ spolynomials2[2]=x^2*y^3+1/2*x*y^4+1/2*x^2*y^2+x*y^3+1/2*x*y^2
```

D.4.32.2 sagbiReduce

Procedure from library `sagbi.lib` (see Section D.4.32 [sagbi_lib], page 1376).

Usage: sagbiReduce(I, A[, tr, mt]); I, A ideals, tr, mt optional integers

Return: ideal of remainders of I after SAGBI reduction by A

Assume: basering is not a qring

Purpose:

The optional argument tr=tailred determines whether tail reduction will be performed.
- If (tailred=0), no tail reduction is done.
- If (tailred<>0), tail reduction is done.

The other optional argument meth determines which method is used for Groebner basis computations.
- If mt=0 (default), the procedure std is used.
- If mt=1, the procedure slimgb is used.

Example:

```
LIB "sagbi.lib";
ring r=0,(x,y,z),dp;
ideal A=x2,2*x2y+y,x3y2;
poly p1=x^5+x2y+y;
poly p2=x^16+x^12*y^5+6*x^8*y^4+x^6+y^4+3;
ideal P=p1,p2;
//------------------------------------------------
//SAGBI reduction of polynomial p1 by algebra A.
//Default call, that is, no tail-reduction is done.
sagbiReduce(p1,A);
↦ x5+x2y+y
//------------------------------------------------
//SAGBI reduction of set of polynomials P by algebra A,
//now tail-reduction is done.
sagbiReduce(P,A,1);
↦ _[1]=x5+1/2y
↦ _[2]=x6y5-8y4
```

D.4.32.3 sagbi

Procedure from library `sagbi.lib` (see Section D.4.32 [sagbi_lib], page 1376).

Usage: sagbi(A[, tr, mt]); A ideal, tr, mt optional integers

Return: ideal, a SAGBI basis for A

Assume: basering is not a qring

Purpose: Computes a SAGBI basis for the subalgebra given by the generators in A.

The optional argument tr=tailred determines whether tail reduction will be performed.
- If (tailred=0), no tail reduction is performed,
- If (tailred<>0), tail reduction is performed.

The other optional argument meth determines which method is used for Groebner basis computations.
- If mt=0 (default), the procedure std is used.
- If mt=1, the procedure slimgb is used.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y,z),dp;
ideal A=x2,y2,xy+y;
//Default call, no tail-reduction is done.
sagbi(A);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=xy+y
↦ _[4]=xy2+1/2y2
//---------------------------------------------
//Call with tail-reduction and method specified.
sagbi(A,1,0);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=xy+y
↦ _[4]=xy2
```

D.4.32.4 sagbiPart

Procedure from library `sagbi.lib` (see Section D.4.32 [sagbi_lib], page 1376).

Usage: sagbiPart(A, k,[tr, mt]); A is an ideal, k, tr and mt are integers

Return: ideal

Assume: basering is not a qring

Purpose: Performs k iterations of the SAGBI construction algorithm for the subalgebra given by the generators given by A.

The optional argument tr=tailred determines if tail reduction will be performed.
- If (tailred=0), no tail reduction is performed,
- If (tailred<>0), tail reduction is performed.
The other optional argument meth determines which method is
used for Groebner basis computations.
- If mt=0 (default), the procedure std is used.
- If mt=1, the procedure slimgb is used.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y,z),dp;
//The following algebra does not have a finite SAGBI basis.
ideal A=x,xy-y2,xy2;
//---------------------------------------------------
//Call with two iterations, no tail-reduction is done.
sagbiPart(A,2);
↦ //SAGBI construction algorithm stopped as it reached the limit of 2 itera\
   tions.
↦ //In general the returned generators are no SAGBI basis for the given alg\
   ebra.
↦ _[1]=x
↦ _[2]=xy-y2
↦ _[3]=xy2
↦ _[4]=2xy3-y4
↦ _[5]=3xy5-y6
↦ _[6]=xy4
//---------------------------------------------------
//Call with three iterations, tail-reduction and method 0.
sagbiPart(A,3,1,0);
↦ //SAGBI construction algorithm stopped as it reached the limit of 3 itera\
   tions.
↦ //In general the returned generators are no SAGBI basis for the given alg\
   ebra.
↦ _[1]=x
↦ _[2]=xy-y2
↦ _[3]=xy2
↦ _[4]=2xy3-y4
↦ _[5]=3xy5-y6
↦ _[6]=xy4
↦ _[7]=5xy9-y10
↦ _[8]=xy8
↦ _[9]=4xy7-y8
↦ _[10]=xy6
```

D.4.32.5 algebraicDependence

Procedure from library `sagbi.lib` (see Section D.4.32 [sagbi_lib], page 1376).

Usage: algebraicDependence(I,it); I an an ideal, it is an integer

Return: ring

Assume: basering is not a qring

Purpose: Returns a ring containing the ideal `algDep`, which contains possibly some algebraic dependencies of the elements of I obtained through `it` iterations of the SAGBI construction algorithms. See the example on how to access these objects.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y),dp;
//The following algebra does not have a finite SAGBI basis.
ideal I=x^2, xy-y2, xy2;
//---------------------------------------------------
//Call with two iterations
def DI = algebraicDependence(I,2);
↦ //AlgDep-1- initialisation and precomputation
↦ //AlgDep-2- call of SAGBI construction algorithm
↦ //SAGBI construction algorithm stopped as it reached the limit of 2 itera\
   tions.
↦ //In general the returned generators are no SAGBI basis for the given alg\
   ebra.
↦ //AlgDep-3- postprocessing of results
setring DI; algDep;
↦ algDep[1]=0
// we see that no dependency has been seen so far
//---------------------------------------------------
//Call with two iterations
setring r; kill DI;
def DI = algebraicDependence(I,3);
↦ //AlgDep-1- initialisation and precomputation
↦ //AlgDep-2- call of SAGBI construction algorithm
↦ //SAGBI construction algorithm stopped as it reached the limit of 3 itera\
   tions.
↦ //In general the returned generators are no SAGBI basis for the given alg\
   ebra.
↦ //AlgDep-3- postprocessing of results
setring DI; algDep;
```

```
↦ algDep[1]=0
map F = DI,x,y,x^2, xy-y2, xy2;
F(algDep); // we see that it is a dependence indeed
↦ _[1]=0
```

D.4.33 sing4ti2_lib

Library: sing4ti2.lib

Purpose: Communication Interface to 4ti2

Authors: Thomas Kahle , kahle@mis.mpg.de
Anne Fruehbis-Krueger, anne@math.uni-hannover.de

Note: This library uses the external program 4ti2 for calculations
and the standard unix tools sed and awk for conversion of
the returned result

Procedures:

D.4.33.1 markov4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.33 [sing4ti2_lib], page 1381).

Usage: markov4ti2(A[,i]);
A=intmat
i=int

Assume: - A is a matrix with integer entries which describes the lattice
as ker(A), if second argument is not present,
as left image Im(A) = {zA, z \in ZZ^k}(!), if second argument is a positive integer
- number of variables of basering equals number of columns of A
(for ker(A)) resp. of rows of A (for Im(A))

Create: files sing4ti2.mat, sing4ti2.lat, sing4ti2.mar in the current
directory (I/O files for communication with 4ti2)

Note: input rules for 4ti2 also apply to input to this procedure
hence ker(A)={x|Ax=0} and Im(A)={xA}

Return: toric ideal specified by Markov basis thereof

Example:

```
LIB "sing4ti2.lib";
ring r=0,(x,y,z),dp;
matrix M[2][3]=0,1,2,2,1,0;
markov4ti2(M);
↦ _[1]=-y2+xz
matrix N[1][3]=1,2,1;
markov4ti2(N,1);
↦ _[1]=xy2z-1
↦ _[2]=xy2z-1
```

D.4.33.2 hilbert4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.33 [sing4ti2_lib], page 1381).

Usage: hilbert4ti2(A[,i]);
A=intmat
i=int

Assume: - A is a matrix with integer entries which describes the lattice
as ker(A), if second argument is not present,
as the left image Im(A) = {zA : z \in ZZ^k}, if second argument is a positive integer
- number of variables of basering equals number of columns of A
(for ker(A)) resp. of rows of A (for Im(A))

Create: temporary files sing4ti2.mat, sing4ti2.lat, sing4ti2.mar
in the current directory (I/O files for communication with 4ti2)

Note: input rules for 4ti2 also apply to input to this procedure
hence ker(A)={x|Ax=0} and Im(A)={xA}

Return: toric ideal specified by Hilbert basis thereof

Example:

```
LIB "sing4ti2.lib";
ring r=0,(x1,x2,x3,x4,x5,x6,x7,x8,x9),dp;
matrix M[7][9]=1,1,1,-1,-1,-1,0,0,0,1,1,1,0,0,0,-1,-1,-1,0,1,1,-1,0,0,-1,0,0,1,0,1,0
hilbert4ti2(M);
↦ _[1]=x1^2*x3*x5*x6^2*x7*x8^2-1
↦ _[2]=x1*x3^2*x4^2*x5*x8^2*x9-1
↦ _[3]=x2^2*x3*x4^2*x5*x7*x9^2-1
↦ _[4]=x1*x2^2*x5*x6^2*x7^2*x9-1
↦ _[5]=x1*x2*x3*x4*x5*x6*x7*x8*x9-1
```

D.4.33.3 graver4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.33 [sing4ti2_lib], page 1381).

Usage: graver4ti2(A[,i]);
A=intmat
i=int

Assume: - A is a matrix with integer entries which describes the lattice
as ker(A), if second argument is not present,
as the left image Im(A) = {zA : z \in ZZ^k}, if second argument is a positive integer
- number of variables of basering equals number of columns of A
(for ker(A)) resp. of rows of A (for Im(A))

Create: temporary files sing4ti2.mat, sing4ti2.lat, sing4ti2.gra
in the current directory (I/O files for communication with 4ti2)

Note: input rules for 4ti2 also apply to input to this procedure
hence ker(A)={x|Ax=0} and Im(A)={xA}

Return: toric ideal specified by Graver basis thereof

Example:

```
LIB "sing4ti2.lib";
ring r=0,(x,y,z,w),dp;
matrix M[2][4]=0,1,2,3,3,2,1,0;
graver4ti2(M);
↦ _[1]=-y2+xz
↦ _[2]=-y3+x2w
↦ _[3]=-yz+xw
↦ _[4]=-z2+yw
↦ _[5]=-z3+xw2
```

D.4.34 symodstd_lib

Library: symodstd.lib

Purpose: Procedures for computing Groebner basis of ideals being invariant under certain variable permutations.

Author: Stefan Steidel, steidel@mathematik.uni-kl.de

Overview: A library for computing the Groebner basis of an ideal in the polynomial ring over the rational numbers, that is invariant under certain permutations of the variables, using the symmetry and modular methods. More precisely let I = <f1,...,fr> be an ideal in Q[x(1),...,x(n)] and sigma a permutation of order k in Sym(n) such that sigma(I) = I. We assume that sigma({f1,...,fr}) = {f1,...,fr}. This can always be obtained by adding sigma(fi) to {f1,...,fr}.
To compute a standard basis of I we apply a modification of the modular version of the standard basis algorithm (improving the calculations in positive characteristic). Therefore we only allow primes p such that p-1 is divisible by k. This guarantees the existence of a k-th primitive root of unity in Z/pZ.

Procedures:

D.4.34.1 genSymId

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: genSymId(I,sigma); I ideal, sigma intvec

Assume: size(sigma) = nvars(basering =: n

Return: ideal J such that sigma(J) = J and J includes I

Note: sigma is a permutation of the variables of the basering, i.e.
sigma: var(i) —-> var(sigma[i]), 1 <= i <= n.

Example:

```
LIB "symodstd.lib";
ring R = 0,(u,v,w,x,y),dp;
intvec pi = 2,3,4,5,1;
ideal I = u2v + x3y - w2;
genSymId(I,pi);
↦ _[1]=x3y+u2v-w2
↦ _[2]=uy3+v2w-x2
↦ _[3]=u3v+w2x-y2
↦ _[4]=v3w+x2y-u2
↦ _[5]=w3x+uy2-v2
```

D.4.34.2 isSymmetric

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: isSymmetric(I,sigma); I ideal, sigma intvec

Assume: size(sigma) = nvars(basering) =: n

Return: 1, if the set of generators of I is invariant under sigma;
0, if the set of generators of I is not invariant under sigma

Note: sigma is a permutation of the variables of the basering, i.e.
sigma: var(i) —-> var(sigma[i]), 1 <= i <= n.

Example:

```
LIB "symodstd.lib";
ring R = 0,x(1..5),dp;
ideal I = cyclic(5);
intvec pi = 2,3,4,5,1;
isSymmetric(I,pi);
↦ 1
intvec tau = 2,5,1,4,3;
isSymmetric(I,tau);
↦ 0
```

D.4.34.3 primRoot

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: primRoot(p,k); p,k integers

Assume: p is a prime and k divides p-1.

Return: int: a k-th primitive root of unity in Z/pZ

Example:

```
LIB "symodstd.lib";
primRoot(181,10);
↦ 56
ring R = 2147482801, x, lp;
number a = primRoot(2147482801,5);
a;
↦ -159774741
a^2;
↦ 140354890
a^3;
↦ 260989846
a^4;
↦ -241569996
a^5;
↦ 1
```

D.4.34.4 eigenvalues

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: eigenvalues(I,sigma); I ideal, sigma intvec

Assume: size(sigma) = nvars(basering) =: n

Return: list of eigenvalues of generators of I under permutation sigma

Note: sigma is a permutation of the variables of the basering, i.e. sigma: var(i) —-> var(sigma[i]), 1 <= i <= n.

Example:

```
LIB "symodstd.lib";
ring R = 11, x(1..5), dp;
poly p1 = x(1)+x(2)+x(3)+x(4)+x(5);
poly p2 = x(1)+4*x(2)+5*x(3)-2*x(4)+3*x(5);
poly p3 = x(1)+5*x(2)+3*x(3)+4*x(4)-2*x(5);
poly p4 = x(1)-2*x(2)+4*x(3)+3*x(4)+5*x(5);
poly p5 = x(1)+3*x(2)-2*x(3)+5*x(4)+4*x(5);
ideal I = p1,p2,p3,p4,p5;
intvec tau = 2,3,4,5,1;
eigenvalues(I,tau);
↦ [1]:
↦    1
↦ [2]:
↦    3
↦ [3]:
↦    -2
↦ [4]:
↦    5
↦ [5]:
↦    4
```

D.4.34.5 symmStd

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: symmStd(I,sigma,#); I ideal, sigma intvec

Assume: size(sigma) = nvars(basering) =: n, basering has an order(sigma)-th primitive root of unity a (if char(basering) > 0) and sigma(I) = I

Return: ideal, a standard basis of I

Note: Assuming that the ideal I is invariant under the variable permutation sigma and the basering has an order(sigma)-th primitive root of unity the procedure uses linear transformation of variables in order to improve standard basis computation.
If char(basering) = 0 all computations are done in the polynomial ring over the smallest field extension that has an order(sigma)-th primitive root of unity.

Example:

```
LIB "symodstd.lib";
ring R = 0, x(1..4), dp;
ideal I = cyclic(4);
I;
↦ I[1]=x(1)+x(2)+x(3)+x(4)
↦ I[2]=x(1)*x(2)+x(2)*x(3)+x(1)*x(4)+x(3)*x(4)
↦ I[3]=x(1)*x(2)*x(3)+x(1)*x(2)*x(4)+x(1)*x(3)*x(4)+x(2)*x(3)*x(4)
↦ I[4]=x(1)*x(2)*x(3)*x(4)-1
intvec pi = 2,3,4,1;
```

```
ideal sI = symmStd(I,pi);
sI;
↦ sI[1]=x(1)+x(2)+x(3)+x(4)
↦ sI[2]=x(2)^2+2*x(2)*x(4)+x(4)^2
↦ sI[3]=x(2)*x(3)^2+x(3)^2*x(4)-x(2)*x(4)^2-x(4)^3
↦ sI[4]=x(2)*x(3)*x(4)^2+x(3)^2*x(4)^2-x(2)*x(4)^3+x(3)*x(4)^3-x(4)^4-1
↦ sI[5]=x(2)*x(4)^4+x(4)^5-x(2)-x(4)
↦ sI[6]=x(3)^3*x(4)^2+x(3)^2*x(4)^3-x(3)-x(4)
↦ sI[7]=x(3)^2*x(4)^4+x(2)*x(3)-x(2)*x(4)+x(3)*x(4)-2*x(4)^2
ring S = 31, (x,y,z), dp;
ideal J;
J[1] = xy-y2+xz;
J[2] = xy+yz-z2;
J[3] = -x2+xz+yz;
intvec tau = 3,1,2;
ideal sJ = symmStd(J,tau);
sJ;
↦ sJ[1]=y2-xz+yz-z2
↦ sJ[2]=xy+yz-z2
↦ sJ[3]=x2-xz-yz
↦ sJ[4]=yz2-z3
↦ sJ[5]=xz2
↦ sJ[6]=z4
```

D.4.34.6 syModStd

Procedure from library `symodstd.lib` (see Section D.4.34 [symodstd_lib], page 1383).

Usage: syModStd(I,sigma); I ideal, sigma intvec

Assume: size(sigma) = nvars(basering) and sigma(I) = I. If size(#) > 0, then # contains either 1, 2 or 4 integers such that
- #[1] is the number of available processors for the computation,
- #[2] is an optional parameter for the exactness of the computation, if #[2] = 1, the procedure computes a standard basis for sure,
- #[3] is the number of primes until the first lifting,
- #[4] is the constant number of primes between two liftings until the computation stops.

Return: ideal, a standard basis of I if no warning appears;

Note: The procedure computes a standard basis of the ideal I (over the rational numbers) by using modular methods and the fact that I is invariant under the variable permutation sigma.
By default the procedure computes a standard basis of I for sure, but if the optional parameter #[2] = 0, it computes a standard basis of I with high probability.
The procedure distinguishes between different variants for the standard basis computation in positive characteristic depending on the ordering of the basering, the parameter #[2] and if the ideal I is homogeneous.
- variant = 1, if I is homogeneous,
- variant = 2, if I is not homogeneous, 1-block-ordering,
- variant = 3, if I is not homogeneous, complicated ordering (lp or > 1 block),
- variant = 4, if I is not homogeneous, ordering lp, dim(I) = 0.

Example:

```
LIB "symodstd.lib";
ring R1 = 0, (x,y,z), dp;
ideal I;
I[1] = -2xyz4+xz5+xz;
I[2] = -2xyz4+yz5+yz;
intvec sigma = 2,1,3;
ideal sI = syModStd(I,sigma);
sI;
↦ sI[1]=x2yz-xy2z
↦ sI[2]=xz5-yz5+xz-yz
↦ sI[3]=xyz4-1/2yz5-1/2yz
↦ sI[4]=y2z5-1/2yz6-xyz+y2z-1/2yz2
ring R2 = 0, x(1..4), dp;
ideal I = cyclic(4);
I;
↦ I[1]=x(1)+x(2)+x(3)+x(4)
↦ I[2]=x(1)*x(2)+x(2)*x(3)+x(1)*x(4)+x(3)*x(4)
↦ I[3]=x(1)*x(2)*x(3)+x(1)*x(2)*x(4)+x(1)*x(3)*x(4)+x(2)*x(3)*x(4)
↦ I[4]=x(1)*x(2)*x(3)*x(4)-1
intvec pi = 2,3,4,1;
ideal sJ1 = syModStd(I,pi,1);
ideal sJ2 = syModStd(I,pi,1,0);
size(reduce(sJ1,sJ2));
↦ 0
size(reduce(sJ2,sJ1));
↦ 0
```

D.4.35 toric_lib

Library: toric.lib

Purpose: Standard Basis of Toric Ideals

Author: Christine Theis, email: ctheis@math.uni-sb.de

Procedures:

D.4.35.1 toric_ideal

Procedure from library `toric.lib` (see Section D.4.35 [toric_lib], page 1387).

Usage: toric_ideal(A,alg); A intmat, alg string
toric_ideal(A,alg,prsv); A intmat, alg string, prsv intvec

Return: ideal: standard basis of the toric ideal of A

Note: These procedures return the standard basis of the toric ideal of A with respect to the term ordering in the current basering. Not all term orderings are supported: The usual global term orderings may be used, but no block orderings combining them.
One may call the procedure with several different algorithms:
- the algorithm of Conti/Traverso using elimination (ect),
- the algorithm of Pottier (pt),
- an algorithm of Bigatti/La Scala/Robbiano (blr),
- the algorithm of Hosten/Sturmfels (hs),
- the algorithm of DiBiase/Urbanke (du).

The argument 'alg' should be the abbreviation for an algorithm as above: ect, pt, blr, hs or du.

If 'alg' is chosen to be 'blr' or 'hs', the algorithm needs a vector with positive coefficients in the row space of A.
If no row of A contains only positive entries, one has to use the second version of toric_ideal which takes such a vector as its third argument.
For the mathematical background, see

Section C.6 [Toric ideals and integer programming], page 781.

Example:

```
LIB "toric.lib";
ring r=0,(x,y,z),dp;
// call with two arguments
intmat A[2][3]=1,1,0,0,1,1;
A;
↦ 1,1,0,
↦ 0,1,1
ideal I=toric_ideal(A,"du");
I;
↦ I[1]=xz-y
I=toric_ideal(A,"blr");
↦    ? The chosen algorithm needs a positive vector in the row space of the\
    matrix.
↦    ? leaving toric.lib::toric_ideal_1 (0)
↦    ? leaving toric.lib::toric_ideal (704)
I;
↦ I[1]=xz-y
// call with three arguments
intvec prsv=1,2,1;
I=toric_ideal(A,"blr",prsv);
I;
↦ I[1]=xz-y
```

See also: Section C.6.1 [Toric ideals], page 781; Section D.4.13 [intprog_lib], page 1118; Section D.4.35.2 [toric_std], page 1388.

D.4.35.2 toric_std

Procedure from library `toric.lib` (see Section D.4.35 [toric_lib], page 1387).

Usage: toric_std(I); I ideal

Return: ideal: standard basis of I

Note: This procedure computes the standard basis of I using a specialized Buchberger algorithm. The generating system by which I is given has to consist of binomials of the form x^u-x^v. There is no real check if I is toric. If I is generated by binomials of the above form, but not toric, toric_std computes an ideal 'between' I and its saturation with respect to all variables.
For the mathematical background, see

Section C.6 [Toric ideals and integer programming], page 781.

Example:

```
LIB "toric.lib";
ring r=0,(x,y,z),wp(3,2,1);
// call with toric ideal (of the matrix A=(1,1,1))
ideal I=x-y,x-z;
ideal J=toric_std(I);
J;
↦ J[1]=y-z
↦ J[2]=x-z
// call with the same ideal, but badly chosen generators:
// 1) not only binomials
I=x-y,2x-y-z;
J=toric_std(I);
↦    ? Generator 2 of the input ideal is no difference of monomials.
↦    ? leaving toric.lib::toric_std (0)
// 2) binomials whose monomials are not relatively prime
I=x-y,xy-yz,y-z;
J=toric_std(I);
↦ Warning: The monomials of generator 2 of the input ideal are not relative\
   ly prime.
J;
↦ J[1]=y-z
↦ J[2]=x-z
// call with a non-toric ideal that seems to be toric
I=x-yz,xy-z;
J=toric_std(I);
J;
↦ J[1]=y2-1
↦ J[2]=x-yz
// comparison with real standard basis and saturation
ideal H=std(I);
H;
↦ H[1]=x-yz
↦ H[2]=y2z-z
LIB "elim.lib";
sat_with_exp(H,xyz);
↦ [1]:
↦    _[1]=x-yz
↦    _[2]=y2-1
↦ [2]:
↦    1
```

See also: Section C.6.1 [Toric ideals], page 781; Section D.4.13 [intprog_lib], page 1118; Section D.4.35.1 [toric_ideal], page 1387; Section D.4.35 [toric_lib], page 1387.

D.5 Algebraic geometry

D.5.1 brillnoether_lib

Library: brillnoether.lib

Purpose: Riemann-Roch spaces of divisors on curves

Authors: I. Stenger: stenger@mathematik.uni-kl.de
Janko Boehm boehm@mathematik.uni-kl.de

Procedures: See also: Section D.10.1 [brnoeth_lib], page 1921.

D.5.1.1 RiemannRochBN

Procedure from library `brillnoether.lib` (see Section D.5.1 [brillnoether_lib], page 1389).

Usage: RiemannRochBN(C,I,J); ideal C, ideal I, ideal J

Assume: C is a homogeneous ideal defining a projective curve. If C is a non-planar curve, then C is assumed to be
nonsingular. This assumption is not checked.
The ideals I and J represent a
a divisor D on C.

Return: A vector space basis of the Riemann-Roch space of D,
stored in a list RRBasis. The list RRBasis contains a list IH and a form F. The vector space basis of L(D)
consists of all rational functions G/F, where G is an element of IH.

Example:

```
LIB "brillnoether.lib";
ring R = 0,(x,y,z),dp;
poly f = y^2+x^2-1;
f = homog(f,z);
ideal C = f;
ideal P1 = x,y-z;
ideal P2 = x^2+y^2,z;
ideal I = intersect(P1^3,P2^2);
ideal P3 = x+z,y;
ideal J = P3^2;
RiemannRochBN(C,I,J);
↦ [1]:
↦    _[1]=65y2z2-81xz3-81z4
↦    _[2]=65xyz2+63xz3+65yz3+63z4
↦    _[3]=65y3z-81xyz2-81yz3
↦    _[4]=65xy2z-81x2z2-81xz3
↦    _[5]=65y4-81xy2z-81y2z2
↦    _[6]=65xy3-81x2yz-81xyz2
↦ [2]:
↦    8xyz2-y2z2-8xz3+2yz3-z4
```

D.5.2 chern_lib

Library: chern.lib

Purpose: Symbolic Computations with Chern classes, Computation of Chern classes

Author: Oleksandr Iena, o.g.yena@gmail.com

Overview: A toolbox for symbolic computations with Chern classes. The Aluffi's algorithms for computation of characteristic classes of algebraic varieties (Segre, Fulton, Chern-Schwartz-MacPherson classes) are implemented as well.

References:

[1] Aluffi, Paolo Computing characteristic classes of projective schemes. Journal of Symbolic Computation, 35 (2003), 3-19. [2] Iena, Oleksandr, On symbolic computations with Chern classes: remarks on the library chern.lib for Singular,

http://hdl.handle.net/10993/22395, 2015.
[3] Lascoux, Alain, Classes de Chern d'un produit tensoriel. C. R. Acad. Sci., Paris, Ser. A 286, 385-387 (1978). [4] Manivel, Laurent Chern classes of tensor products, arXiv 1012.0014, 2010.

Procedures:

D.5.2.1 symm

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: symm(l [,n]); l a list of polynomials, n integer

Return: list of polynomials

Purpose: computes the list of elementary symmetric functions in the entries of l

Note: makes sense only for a list of polynomials

Example:

```
LIB "chern.lib";
// elementary symmetric functions in x, y, z:
ring r = 0, (x, y, z), dp;
list l=(x, y, z);
print(symm(l));
↦ [1]:
↦   x+y+z
↦ [2]:
↦   xy+xz+yz
↦ [3]:
↦   xyz
//now let us compute only the first two symmetric polynomials in a(1), ... , a(10)
ring q= 0,(a(1..10)), dp;
list l=a(1..10);
print(symm(l, 2));
↦ [1]:
↦   a(1)+a(2)+a(3)+a(4)+a(5)+a(6)+a(7)+a(8)+a(9)+a(10)
↦ [2]:
↦   a(1)*a(2)+a(1)*a(3)+a(2)*a(3)+a(1)*a(4)+a(2)*a(4)+a(3)*a(4)+a(1)*a(5)+\
  a(2)*a(5)+a(3)*a(5)+a(4)*a(5)+a(1)*a(6)+a(2)*a(6)+a(3)*a(6)+a(4)*a(6)+a(5\
  )*a(6)+a(1)*a(7)+a(2)*a(7)+a(3)*a(7)+a(4)*a(7)+a(5)*a(7)+a(6)*a(7)+a(1)*a\
  (8)+a(2)*a(8)+a(3)*a(8)+a(4)*a(8)+a(5)*a(8)+a(6)*a(8)+a(7)*a(8)+a(1)*a(9)\
  +a(2)*a(9)+a(3)*a(9)+a(4)*a(9)+a(5)*a(9)+a(6)*a(9)+a(7)*a(9)+a(8)*a(9)+a(\
  1)*a(10)+a(2)*a(10)+a(3)*a(10)+a(4)*a(10)+a(5)*a(10)+a(6)*a(10)+a(7)*a(10\
  )+a(8)*a(10)+a(9)*a(10)
```

D.5.2.2 symNsym

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: symNsym(f, c); f polynomial; c list of polynomials

Return: list with 2 poly entries

Purpose: computes a symmetric and a non-symmetric part of f
in terms of the elementary symmetric functions from c as well a non-symmetric remainder

Note: constants are considered symmetric

Example:

```
LIB "chern.lib";
ring r=0, (x,y,z, c(1..3)), dp;
list l=c(1..3);
// The symmetric part of f = 3x2 + 3y2 + 3z2 + 7xyz + y
// in terms of the elementary symmetric functions c(1), c(2), c(3)
// and the remainder
poly f = 3x2 + 3y2 + 3z2 + 7xyz + y;
print( symNsym(f, l) );
↦ [1]:
↦    3*c(1)^2-6*c(2)+7*c(3)
↦ [2]:
↦    y
// Take a symmetric polynomial in variables x and z
f=x2+xz+z2;
// Express it in terms of the elementary the symmetric functions
print( symNsym(f, l)[1]);
↦ c(1)^2-c(2)
```

D.5.2.3 CompleteHomog

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: CompleteHomog(N, c); N integer, c list of polynomials

Return: list of polynomials

Purpose: computes the list of the complete homogeneous symmetric polynomials in terms of the elementary symmetric polynomials (entries of c)

Note:

Example:

```
LIB "chern.lib";
ring r = 0, (x(1..3)), dp;
list l=x(1..3);
//Complete homogeneous symmetric polynomials up to degree 3 in variables x(1), x(2),
print( CompleteHomog(3, l) );
↦ [1]:
↦    1
↦ [2]:
↦    x(1)
↦ [3]:
↦    x(1)^2-x(2)
↦ [4]:
↦    x(1)^3-2*x(1)*x(2)+x(3)
```

D.5.2.4 segre

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: segre(c[, N]); c list of polynomials, N integer

Return: list of polynomials

Purpose: computes the list of the Segre classes up to degree N in terms of the Chern classes from c

Note:

Example:

```
LIB "chern.lib";
ring r = 0, (c(1..3)), dp;
list l=c(1..3);
//Segre classes up to degree 5 in Chern classes c(1), c(2), c(3)
print( segre(l, 5) );
↦ [1]:
↦    -c(1)
↦ [2]:
↦    c(1)^2-c(2)
↦ [3]:
↦    -c(1)^3+2*c(1)*c(2)-c(3)
↦ [4]:
↦    c(1)^4-3*c(1)^2*c(2)+c(2)^2+2*c(1)*c(3)
↦ [5]:
↦    -c(1)^5+4*c(1)^3*c(2)-3*c(1)*c(2)^2-3*c(1)^2*c(3)+2*c(2)*c(3)
```

D.5.2.5 chern

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chern(s); s list of polynomials

Return: list of polynomials

Purpose: computes the list of the Chern classes up to degree N in terms of the Segre classes from s

Note:

Example:

```
LIB "chern.lib";
ring r = 0, (s(1..3)), dp;
list l=s(1..3);
// Chern classes in Segre classes s(1), s(2), s(3)
print( chern(l) );
↦ [1]:
↦    -s(1)
↦ [2]:
↦    s(1)^2-s(2)
↦ [3]:
↦    -s(1)^3+2*s(1)*s(2)-s(3)
// This procedure is inverse to segre(...). Indeed:
print( segre(chern(l), 3) );
↦ [1]:
↦    s(1)
↦ [2]:
↦    s(2)
↦ [3]:
↦    s(3)
```

D.5.2.6 chNum

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chNum(N, c); N integer, c list

Return: list

Purpose: computes the Chern numbers of a vector bundle with Chern classes c on a complex manifold (variety) of dimension N,
the zeroes corresponding to the higher zero Chern classes are ignored

Note: computes basically the partitions of N
in summands not greater than the length of c

Example:

```
LIB "chern.lib";
ring r = 0, (c(1..2)), dp;
list l=c(1..2);
// Let c(1) be a variable of degree 1, let c(2) be a variable of degree 2.
// The monomials in c(1) and c(2) of weighted degree 5 are:
print( chNum( 5, l ) );
↦ [1]:
↦    c(1)^5
↦ [2]:
↦    c(1)^3*c(2)
↦ [3]:
↦    c(1)*c(2)^2
// Compare the result to the output of chNumbers(...):
print( chNumbers(5, l) );
↦ [1]:
↦    c(1)^5
↦ [2]:
↦    c(1)^3*c(2)
↦ [3]:
↦    c(1)*c(2)^2
↦ [4]:
↦    0
↦ [5]:
↦    0
↦ [6]:
↦    0
↦ [7]:
↦    0
```

D.5.2.7 chNumbers

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chNumbers(r, c); r integer, c list

Return: list

Purpose: computes the Chern numbers of a vector bundle with Chern classes c on a complex manifold (variety) of dimension r

Note: computes basically the partitions of r

Example:

```
LIB "chern.lib";
ring r = 0, (c(1..3)), dp;
list l=c(1..3);
// The Chern numbers of a vector bundle with Chern classes c(1), c(2), c(3)
// on a 3-fold:
print( chNumbers( 3, l ) );
↦ [1]:
↦    c(1)^3
↦ [2]:
↦    c(1)*c(2)
↦ [3]:
↦    c(3)
// If the highest Chern class is zero, the Chern numbers are:
l=c(1..2);
print( chNumbers( 3, l ) );
↦ [1]:
↦    c(1)^3
↦ [2]:
↦    c(1)*c(2)
↦ [3]:
↦    0
// Compare this to the output of chNum(...):
print( chNum( 3, l ) );
↦ [1]:
↦    c(1)^3
↦ [2]:
↦    c(1)*c(2)
```

D.5.2.8 sum_of_powers

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: sum_of_powers(k, l); k non-negative integer, l list of polynomials

Return: polynomial

Purpose: computes the sum of k-th powers of the entries of l

Note: returns 0 if k is negative

Example:

```
LIB "chern.lib";
ring r = 0, (x, y, z), dp;
list l=x, y, z;
//sum of 7-th powers of x, y, z
print( sum_of_powers(7, l) );
↦ x7+y7+z7
```

D.5.2.9 powSumSym

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: powSumSym(l [,N]); l a list of polynomials, N integer

Return: list of polynomials

Purpose: computes the expressions for the sums of powers [up to degree N] in terms of the elementary symmetric polynomials (entries of l),

Note: returns the terms of the Chern character
multiplied by the corresponding factorials

Example:

```
LIB "chern.lib";
// the expressions of the first 3 sums of powers of 3 variables a(1), a(2), a(3)
// in terms of the elementary symmetric polynomials c(1), c(2), c(3):
ring r = 0, (c(1..3)), dp;
list l=(c(1..3));
print(powSumSym(l));
↦ [1]:
↦    c(1)
↦ [2]:
↦    c(1)^2-2*c(2)
↦ [3]:
↦    c(1)^3-3*c(1)*c(2)+3*c(3)
// The first 5 sums in the same situation
print(powSumSym(l, 5));
↦ [1]:
↦    c(1)
↦ [2]:
↦    c(1)^2-2*c(2)
↦ [3]:
↦    c(1)^3-3*c(1)*c(2)+3*c(3)
↦ [4]:
↦    c(1)^4-4*c(1)^2*c(2)+2*c(2)^2+4*c(1)*c(3)
↦ [5]:
↦    c(1)^5-5*c(1)^3*c(2)+5*c(1)*c(2)^2+5*c(1)^2*c(3)-5*c(2)*c(3)
```

D.5.2.10 chAll

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chAll(l [,N]); l a list of polynomials, N integer

Return: list of polynomials

Purpose: computes the list of terms of positive degree [up to degree N] of the Chern character, where the entries of l are considered as the Chern classes

Note: makes sense only for a list of polynomials

Example:

```
LIB "chern.lib";
// Chern character (terms of degree 1, 2, 3)
// corresponding to the Chern classes c(1), c(2), c(3):
ring r = 0, (c(1..3)), dp;
list l=(c(1..3));
print(chAll(l));
↦ [1]:
↦    c(1)
↦ [2]:
↦    1/2*c(1)^2-c(2)
↦ [3]:
↦    1/6*c(1)^3-1/2*c(1)*c(2)+1/2*c(3)
// terms up to degree 5 in the same situation
print(chAll(l, 5));
↦ [1]:
↦    c(1)
↦ [2]:
↦    1/2*c(1)^2-c(2)
↦ [3]:
↦    1/6*c(1)^3-1/2*c(1)*c(2)+1/2*c(3)
↦ [4]:
↦    1/24*c(1)^4-1/6*c(1)^2*c(2)+1/12*c(2)^2+1/6*c(1)*c(3)
↦ [5]:
↦    1/120*c(1)^5-1/24*c(1)^3*c(2)+1/24*c(1)*c(2)^2+1/24*c(1)^2*c(3)-1/24*c\
   (2)*c(3)
```

D.5.2.11 chAllInv

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chAllInv(l); l a list of polynomials

Return: list of polynomials

Purpose: procedure inverse to chAll(), computes the list of Chern classes from the list of terms of positive degree of the Chern character

Note: makes sense only for a list of polynomials

Example:

```
LIB "chern.lib";
// first 3 Chern classes in terms of the first 3 terms
// of the Chern character Chern  ch(1), ch(2), ch(3):
ring r = 0, (ch(1..3)), dp;
list l=(ch(1..3));
print(chAllInv(l));
↦ [1]:
↦    ch(1)
↦ [2]:
↦    1/2*ch(1)^2-ch(2)
↦ [3]:
↦    1/6*ch(1)^3-ch(1)*ch(2)+2*ch(3)
// let's see that chAllInv() is inverse to chAll()
print( chAll( chAllInv(l) ) );
↦ [1]:
↦    ch(1)
↦ [2]:
↦    ch(2)
↦ [3]:
↦    ch(3)
```

D.5.2.12 chHE

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chHE(c); c list of polynomials

Return: polynomial

Purpose: computes the highest relevant term of the Chern character

Note: uses the elimination and is extremely inefficient,
is included just for comparison with chAll(c)

Example:

```
LIB "chern.lib";
ring r = 0, (c(1..3)), dp;
list l=c(1..3);
//the third degree term of the Chern character
print( chHE(l) );
↦ 1/6*c(1)^3-1/2*c(1)*c(2)+1/2*c(3)
```

D.5.2.13 ChernRootsSum

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsSum(a, b); a, b lists of polynomials

Return: list of polynomials

Purpose: computes the Chern roots of the direct (Whitney) sum of a vector bundle with Chern roots a and a vector bundle with Chern roots b

Note:

Example:

```
LIB "chern.lib";
ring r = 0, (a(1..3), b(1..2)), dp;
// assume a(1), a(2), a(3) are the Chern roots of a vector bundle E
// assume b(1), b(2) are the Chern roots of a vector bundle F
list l=a(1..3);
list L=b(1..2);
// the Chern roots of their direct sum is
print( ChernRootsSum(l, L) );
↦ [1]:
↦    a(1)
↦ [2]:
↦    a(2)
↦ [3]:
↦    a(3)
↦ [4]:
↦    b(1)
↦ [5]:
↦    b(2)
```

D.5.2.14 chSum

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chSum(c, C); c, C lists of polynomials

Return: list of polynomials

Purpose: computes the Chern classes of a direct sum of two vector bundles

Note:

Example:

```
     LIB "chern.lib";
     ring r = 0, (c(1..3), C(1..2)), dp;
     // Let E be a vector bundle with Chern classes c(1), c(2), c(3).
     // Let F be a vector bundle with Chern classes C(1), C(2).
     list l=c(1..3);
     list L=C(1..2);
     // Then the Chern classes of their direct sum are
     print( chSum(l, L) );
     ↦ [1]:
     ↦    c(1)+C(1)
     ↦ [2]:
     ↦    c(1)*C(1)+c(2)+C(2)
     ↦ [3]:
     ↦    c(2)*C(1)+c(1)*C(2)+c(3)
     ↦ [4]:
     ↦    c(3)*C(1)+c(2)*C(2)
     ↦ [5]:
     ↦    c(3)*C(2)
```

D.5.2.15 ChernRootsDual

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsDual(l); l a list of polynomials

Return: list of polynomials

Purpose: computes the Chern roots of the dual vector bundle
of a vector bundle with Chern roots from l

Note:

Example:

```
     LIB "chern.lib";
     ring r = 0, (a(1..3)), dp;
     // assume a(1), a(2), a(3) are the Chern roots of a vector bundle
     list l=a(1..3);
     // the Chern roots of the dual vector bundle
     print( ChernRootsDual(l) );
     ↦ [1]:
     ↦    -a(1)
     ↦ [2]:
     ↦    -a(2)
     ↦ [3]:
     ↦    -a(3)
```

D.5.2.16 chDual

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chDual(c); c list of polynomials

Return: list of polynomials

Purpose: computes the list of Chern classes of the dual vector bundle

Note:

Example:

```
LIB "chern.lib";
// Chern classes of a vector bundle that is dual to a vector bundle
// with Chern classes c(1), c(2), c(3)
ring r=0, (c(1..3)), dp;
list l=c(1..3);
print(chDual(l));
↦ [1]:
↦    -c(1)
↦ [2]:
↦    c(2)
↦ [3]:
↦    -c(3)
```

D.5.2.17 ChernRootsProd

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsProd(a, b); a, b lists of polynomials

Return: list of polynomials

Purpose: computes the Chern roots of the tensor product of a vector bundle with Chern roots a and a vector bundles with Chern roots b

Note:

Example:

```
LIB "chern.lib";
ring r=0, (a(1..2), b(1..3)), dp;
list l=a(1..2);
list L=b(1..3);
// Chern roots of the tensor product of a vector bundle with Chern roots a(1), a(2)
// and a vector bundle with Chern roots b(1), b(2), b(3)
print(ChernRootsProd(l, L));
↦ [1]:
↦    a(1)+b(1)
↦ [2]:
↦    a(1)+b(2)
↦ [3]:
↦    a(1)+b(3)
↦ [4]:
↦    a(2)+b(1)
↦ [5]:
↦    a(2)+b(2)
↦ [6]:
↦    a(2)+b(3)
```

D.5.2.18 chProd

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chProd(r, c, R, C [, N]); r, R polynomials (integers); c, C lists of polynomials, N integer

Return: list of polynomials

Purpose: computes the list of Chern classes of the product of two vector bundles in terms of their ranks and Chern clases [up to degree N]

Note:

Example:

```
LIB "chern.lib";
ring H = 0, ( r, R, c(1..3), C(1..2) ), dp;
list l=c(1..3);
list L=C(1..2);
// the Chern classes of the tensor product of a vector bundle E of rank 3
// with Chern classes c(1), c(2), c(3)
// and a vector bundle F of rank 2 with Chern classes C(1) and C(2):
print( chProd(3, l, 2, L) );
↦ [1]:
↦    2*c(1)+3*C(1)
↦ [2]:
↦    c(1)^2+5*c(1)*C(1)+3*C(1)^2+2*c(2)+3*C(2)
↦ [3]:
↦    2*c(1)^2*C(1)+4*c(1)*C(1)^2+C(1)^3+2*c(1)*c(2)+4*c(2)*C(1)+4*c(1)*C(2)\
  +6*C(1)*C(2)+2*c(3)
↦ [4]:
↦    c(1)^2*C(1)^2+c(1)*C(1)^3+3*c(1)*c(2)*C(1)+3*c(2)*C(1)^2+2*c(1)^2*C(2)\
  +6*c(1)*C(1)*C(2)+3*C(1)^2*C(2)+c(2)^2+2*c(1)*c(3)+3*c(3)*C(1)+3*C(2)^2
↦ [5]:
↦    c(1)*c(2)*C(1)^2+c(2)*C(1)^3+2*c(1)^2*C(1)*C(2)+2*c(1)*C(1)^2*C(2)+c(2\
  )^2*C(1)+2*c(1)*c(3)*C(1)+3*c(3)*C(1)^2+2*c(1)*c(2)*C(2)+2*c(1)*C(2)^2+3*\
  C(1)*C(2)^2+2*c(2)*c(3)-6*c(3)*C(2)
↦ [6]:
↦    c(1)*c(3)*C(1)^2+c(3)*C(1)^3+c(1)*c(2)*C(1)*C(2)+c(2)*C(1)^2*C(2)+c(1)\
  ^2*C(2)^2+c(1)*C(1)*C(2)^2+c(2)*c(3)*C(1)+c(2)^2*C(2)-2*c(1)*c(3)*C(2)-3*\
  c(3)*C(1)*C(2)-2*c(2)*C(2)^2+C(2)^3+c(3)^2
// the first two Chern classes of the tensor product
// of a vector bundle E of rank r with Chern classes c(1) and c(2)
// and a vector bundle G of rank R with Chern classes C(1) and C(2)
// this gives the Chern classes of a tensor product on a complex surface
l=c(1..2);
L=C(1..2);
print( chProd(r, l, R, L, 2 ) );
↦ [1]:
↦    R*c(1)+r*C(1)
↦ [2]:
↦    1/2*R^2*c(1)^2+r*R*c(1)*C(1)+1/2*r^2*C(1)^2-1/2*R*c(1)^2-1/2*r*C(1)^2+\
  R*c(2)-c(1)*C(1)+r*C(2)
```

D.5.2.19 chProdE

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chProdE(c, C); c, C lists of polynomials

Return: list of polynomials

Purpose: computes the list of Chern classes of the product
of two vector bundles in terms of their Chern clases

Note: makes sense only for (lists of) polynomials;
uses elimination, hence very inefficient;
included only for comparison with chProd(...)

Example:

```
LIB "chern.lib";
ring H = 0, ( c(1..3), C(1..2) ), dp;
list l=c(1..3);
list L=C(1..2);
// the Chern classes of the tensor product of a vector bundle E of rank 3
// with Chern classes c(1), c(2), c(3)
// and a vector bundle F of rank 2 with Chern classes C(1) and C(2):
print( chProdE(l,  L) );
↦ [1]:
↦    2*c(1)+3*C(1)
↦ [2]:
↦    c(1)^2+5*c(1)*C(1)+3*C(1)^2+2*c(2)+3*C(2)
↦ [3]:
↦    2*c(1)^2*C(1)+4*c(1)*C(1)^2+C(1)^3+2*c(1)*c(2)+4*c(2)*C(1)+4*c(1)*C(2)\
   +6*C(1)*C(2)+2*c(3)
↦ [4]:
↦    c(1)^2*C(1)^2+c(1)*C(1)^3+3*c(1)*c(2)*C(1)+3*c(2)*C(1)^2+2*c(1)^2*C(2)\
   +6*c(1)*C(1)*C(2)+3*C(1)^2*C(2)+c(2)^2+2*c(1)*c(3)+3*c(3)*C(1)+3*C(2)^2
↦ [5]:
↦    c(1)*c(2)*C(1)^2+c(2)*C(1)^3+2*c(1)^2*C(1)*C(2)+2*c(1)*C(1)^2*C(2)+c(2\
   )^2*C(1)+2*c(1)*c(3)*C(1)+3*c(3)*C(1)^2+2*c(1)*c(2)*C(2)+2*c(1)*C(2)^2+3*\
   C(1)*C(2)^2+2*c(2)*c(3)-6*c(3)*C(2)
↦ [6]:
↦    c(1)*c(3)*C(1)^2+c(3)*C(1)^3+c(1)*c(2)*C(1)*C(2)+c(2)*C(1)^2*C(2)+c(1)\
   ^2*C(2)^2+c(1)*C(1)*C(2)^2+c(2)*c(3)*C(1)+c(2)^2*C(2)-2*c(1)*c(3)*C(2)-3*\
   c(3)*C(1)*C(2)-2*c(2)*C(2)^2+C(2)^3+c(3)^2
```

D.5.2.20 chProdL

Procedure from library chern.lib (see Section D.5.2 [chern_lib], page 1390).

Usage: chProdL(r, c, R, C); r, R integers; c, C lists of polynomials

Return: list

Purpose: computes the list of Chern classes of the product of two vector bundles in terms of their Chern clases

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
// The Chern classes of the tensor product of a vector bundle of rank 3
// with Chern classes c(1), c(2), c(3) and a vector bundle of rank 1 with
// Chern class C(1)
ring r = 0, ( c(1..3), C(1)), dp;
list c=c(1..3);
list C=C(1);
print( chProdL(3,c,1,C) );
↦ [1]:
↦    c(1)+3*C(1)
↦ [2]:
↦    2*c(1)*C(1)+3*C(1)^2+c(2)
↦ [3]:
↦    c(1)*C(1)^2+C(1)^3+c(2)*C(1)+c(3)
```

D.5.2.21 chProdLP

Procedure from library chern.lib (see Section D.5.2 [chern_lib], page 1390).

Usage: chProdLP(r, c, R, C); r, R integers; c, C lists of polynomials

Return: polynomial

Purpose: computes the total Chern class of the product of two vector bundles in terms of their ranks and Chern clases

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
// The total Chern class of the tensor product of a vector bundle of rank 3
// with Chern classes c(1), c(2), c(3) and a vector bundle of rank 1 with
// Chern class C(1)
ring r = 0, ( c(1..3), C(1)), ws(1,2,3, 1);
list c=c(1..3);
list C=C(1);
print( chProdLP(3,c,1,C) );
↦ 1+c(1)+3*C(1)+c(2)+2*c(1)*C(1)+3*C(1)^2+c(3)+c(2)*C(1)+c(1)*C(1)^2+C(1)^3
```

D.5.2.22 chProdM

Procedure from library chern.lib (see Section D.5.2 [chern_lib], page 1390).

Usage: chProdM(r, c, R, C); r, R integers; c, C lists of polynomials

Return: list

Purpose: computes the list of Chern classes of the product of two vector bundles in terms of their Chern clases

Note: Implementation of the formula of Manivel

Example:

```
LIB "chern.lib";
// The Chern classes of the tensor product of a vector bundle of rank 3
// with Chern classes c(1), c(2), c(3) and a vector bundle of rank 1 with
// Chern class C(1)
ring r = 0, ( c(1..3), C(1)), dp;
list c=c(1..3);
list C=C(1);
print( chProdM(3,c,1,C) );
↦ [1]:
```

```
↦    c(1)+3*C(1)
↦ [2]:
↦    2*c(1)*C(1)+3*C(1)^2+c(2)
↦ [3]:
↦    c(1)*C(1)^2+C(1)^3+c(2)*C(1)+c(3)
```

D.5.2.23 chProdMP

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chProdMP(r, c, R, C); r, R integers; c, C lists of polynomials

Return: polynomial

Purpose: computes the total Chern class of the product of two vector bundles in terms of their ranks and Chern clases

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
// The total Chern class of the tensor product of a vector bundle of rank 3
// with Chern classes c(1), c(2), c(3) and a vector bundle of rank 1 with
// Chern class C(1)
ring r = 0, ( c(1..3), C(1)), ws(1,2,3, 1);
list c=c(1..3);
list C=C(1);
print( chProdMP(3,c,1,C) );
↦ 1+c(1)+3*C(1)+c(2)+2*c(1)*C(1)+3*C(1)^2+c(3)+c(2)*C(1)+c(1)*C(1)^2+C(1)^3
```

D.5.2.24 ChernRootsHom

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsHom(a, b); a, b lists of polynomials

Return: list of polynomials

Purpose: for a vector bundle E with Chern roots a and a vector bundle F with Chern roots b, computes the Chern roots of Hom(E, F)

Note:

Example:

```
LIB "chern.lib";
ring r=0, (a(1..2), b(1..3)), dp;
list l=a(1..2);
list L=b(1..3);
// Let E be a vector bundle with Chern roots a(1). a(2),
// let F be a vector bundle with CHern roots b(1), b(2), b(3).
// Then the Chern roots of Hom(E, F) are
print(ChernRootsHom(l, L));
↦ [1]:
↦    -a(1)+b(1)
↦ [2]:
↦    -a(1)+b(2)
↦ [3]:
↦    -a(1)+b(3)
↦ [4]:
↦    -a(2)+b(1)
↦ [5]:
↦    -a(2)+b(2)
↦ [6]:
↦    -a(2)+b(3)
```

D.5.2.25 chHom

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chHom(r, c, R, C [, N]); r, R polynomials (integers); c, C lists of polynomials, N integer

Return: list of polynomials

Purpose: computes [up to degree N] the list of Chern classes of the vector bundle Hom(E, F) in terms of the ranks and the Chern classes of E and F

Note:

Example:

```
LIB "chern.lib";
ring H = 0, ( r, R, c(1..3), C(1..2) ), dp;
list l=c(1..3);
list L=C(1..2);
// the Chern classes of Hom(E, F) for a vector bundle E of rank 3
// with Chern classes c(1), c(2), c(3)
// and a vector bundle F of rank 2 with Chern classes C(1) and C(2):
print( chHom(3, l, 2, L) );
↦ [1]:
↦    -2*c(1)+3*C(1)
↦ [2]:
↦    c(1)^2-5*c(1)*C(1)+3*C(1)^2+2*c(2)+3*C(2)
↦ [3]:
↦    2*c(1)^2*C(1)-4*c(1)*C(1)^2+C(1)^3-2*c(1)*c(2)+4*c(2)*C(1)-4*c(1)*C(2)\
   +6*C(1)*C(2)-2*c(3)
↦ [4]:
↦    c(1)^2*C(1)^2-c(1)*C(1)^3-3*c(1)*c(2)*C(1)+3*c(2)*C(1)^2+2*c(1)^2*C(2)\
   -6*c(1)*C(1)*C(2)+3*C(1)^2*C(2)+c(2)^2+2*c(1)*c(3)-3*c(3)*C(1)+3*C(2)^2
↦ [5]:
↦    -c(1)*c(2)*C(1)^2+c(2)*C(1)^3+2*c(1)^2*C(1)*C(2)-2*c(1)*C(1)^2*C(2)+c(\
   2)^2*C(1)+2*c(1)*c(3)*C(1)-3*c(3)*C(1)^2-2*c(1)*c(2)*C(2)-2*c(1)*C(2)^2+3\
   *C(1)*C(2)^2-2*c(2)*c(3)+6*c(3)*C(2)
↦ [6]:
↦    c(1)*c(3)*C(1)^2-c(3)*C(1)^3-c(1)*c(2)*C(1)*C(2)+c(2)*C(1)^2*C(2)+c(1)\
   ^2*C(2)^2-c(1)*C(1)*C(2)^2-c(2)*c(3)*C(1)+c(2)^2*C(2)-2*c(1)*c(3)*C(2)+3*\
   c(3)*C(1)*C(2)-2*c(2)*C(2)^2+C(2)^3+c(3)^2
// the first two Chern classes of Hom(E, F) for a vector bundle E of rank r
// with Chern classes c(1) and c(2)
// and a vector bundle G of rank R with Chern classes C(1) and C(2)
// this gives the Chern classes of a tensor product on a complex surface
l=c(1..2);
L=C(1..2);
```

```
print( chHom(r, l, R, L, 2 ) );
↦ [1]:
↦    -R*c(1)+r*C(1)
↦ [2]:
↦    1/2*R^2*c(1)^2-r*R*c(1)*C(1)+1/2*r^2*C(1)^2-1/2*R*c(1)^2-1/2*r*C(1)^2+\
   R*c(2)+c(1)*C(1)+r*C(2)
```

D.5.2.26 ChernRootsSymm

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsSymm(m, l); m integer, l a list of polynomials

Return: list of polynomials

Purpose: computes the Chern roots of m-th symmetric power
of a vector bundle with Chern roots from l

Note:

Example:

```
LIB "chern.lib";
ring r=0, (a(1..3)), dp;
list l=a(1..3);
// the Chern roots of the second symmetric power of a vector bundle
// with Chern  roots a(1), a(2), a(3)
print( ChernRootsSymm(2, l) );
↦ [1]:
↦    2*a(1)
↦ [2]:
↦    a(1)+a(2)
↦ [3]:
↦    a(1)+a(3)
↦ [4]:
↦    2*a(2)
↦ [5]:
↦    a(2)+a(3)
↦ [6]:
↦    2*a(3)
```

D.5.2.27 ChernRootsWedge

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: ChernRootsWedge(m, l); m integer, l a list of polynomials

Return: list of polynomials

Purpose: computes the Chern roots of m-th exterior power
of a vector bundle with Chern roots from l

Note: makes sense only for list of polynomials

Example:

```
LIB "chern.lib";
ring r=0, (a(1..3)), dp;
list l=a(1..3);
```

```
// the Chern roots of the second exterior power of a vector bundle
// with Chern  roots a(1), a(2), a(3)
print( ChernRootsWedge(2, l) );
↦ [1]:
↦    a(2)+a(3)
↦ [2]:
↦    a(1)+a(3)
↦ [3]:
↦    a(1)+a(2)
```

D.5.2.28 chSymm

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chSymm(k, r, c[, pos]); k, r integers, c list of polynomials, pos list of integers

Return: list with entries: int N, list of polynomials l

Purpose: computes the rank and the Chern classes of the symmetric power of a vector bundle

Note: for the second symmetric power chSymm2L(...) could be faster

Example:

```
LIB "chern.lib";
ring r=0, (c(1..5)), dp;
list l=c(1..5);
// the rank and the Chern classes of the second symmetric power of a vector bundle o
print( chSymm(2, 3, l) );
↦ [1]:
↦    6
↦ [2]:
↦    [1]:
↦       4*c(1)
↦    [2]:
↦       5*c(1)^2+5*c(2)
↦    [3]:
↦       2*c(1)^3+11*c(1)*c(2)+7*c(3)
↦    [4]:
↦       6*c(1)^2*c(2)+4*c(2)^2+14*c(1)*c(3)
↦    [5]:
↦       4*c(1)*c(2)^2+8*c(1)^2*c(3)+4*c(2)*c(3)
↦    [6]:
↦       8*c(1)*c(2)*c(3)-8*c(3)^2
// the rank and the first 3 Chern classes
// of the second symmetric power of a vector bundle of rank 5
print( chSymm(2, 5, l, 1, 2, 3) );
↦ [1]:
↦    15
↦ [2]:
↦    [1]:
↦       6*c(1)
↦    [2]:
↦       14*c(1)^2+7*c(2)
↦    [3]:
↦       16*c(1)^3+31*c(1)*c(2)+9*c(3)
```

D.5.2.29 chSymm2L

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chSymm2L(r, c); r integer, c list of polynomials

Return: list of polynomials

Purpose: computes the Chern classes of the second symmetric power of a vector bundle

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
ring r=0, (c(1..2)), dp;
list l=c(1..2);
// the Chern classes of the second symmetric power of a vector bundle of rank 2
print( chSymm2L(2, l));
↦ [1]:
↦    3
↦ [2]:
↦    [1]:
↦       3*c(1)
↦    [2]:
↦       2*c(1)^2+4*c(2)
↦    [3]:
↦       4*c(1)*c(2)
```

D.5.2.30 chSymm2LP

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chSymm2LP(r, c); r integer, c list of polynomials

Return: poly

Purpose: computes the total Chern class of the second symmetric power of a vector bundle

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
ring r=0, (c(1..2)), ws(1, 2);
list l=c(1..2);
// the total Chern class of the second symmetric power of a vector bundle of rank 2
print( chSymm2LP(2, l));
↦ 1+3*c(1)+2*c(1)^2+4*c(2)+4*c(1)*c(2)
```

D.5.2.31 chWedge

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chWedge(k, r, c [,pos]); k, r integers, c list of polynomials, pos list of integers

Return: list with entries: int N, list of polynomials l

Purpose: computes the rank and the Chern classes of the exterior power of a vector bundle

Note: for the second exterior power chWedge2L(...) could be faster

Example:

```
LIB "chern.lib";
ring r=0, (c(1..5)), dp;
list l=c(1..5);
// the rank and the Chern classes of the second exterior power of a vector bundle of
print( chWedge(2, 3, l) );
↦ [1]:
↦    3
↦ [2]:
↦    [1]:
↦       2*c(1)
↦    [2]:
↦       c(1)^2+c(2)
↦    [3]:
↦       c(1)*c(2)-c(3)
// the rank and the first 3 Chern classes
// of the fourth exterior power of a vector bundle of rank 5
print( chWedge(4, 5, l, 1, 2, 3) );
↦ [1]:
↦    5
↦ [2]:
↦    [1]:
↦       4*c(1)
↦    [2]:
↦       6*c(1)^2+c(2)
↦    [3]:
↦       4*c(1)^3+3*c(1)*c(2)-c(3)
```

D.5.2.32 chWedge2L

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chWedge2L(r, c); r integer, c list of polynomials

Return: list of polynomials

Purpose: computes the Chern classes of the second exterior power of a vector bundle

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
ring r=0, (c(1..3)), dp;
list l=c(1..3);
// the Chern classes of the second exterior power of a vector bundle of rank 3
print(chWedge2L(3, l));
↦ [1]:
↦    3
↦ [2]:
↦    [1]:
↦       2*c(1)
↦    [2]:
↦       c(1)^2+c(2)
↦     [3]:
↦        c(1)*c(2)-c(3)
```

D.5.2.33 chWedge2LP

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chWedge2LP(r, c); r integer, c list of polynomials

Return: poly

Purpose: computes the total Chern class of the second exterior power of a vector bundle

Note: Implementation of the formula of Lascoux, the Schur polynomials are computed using the second Jacobi-Trudi formula (in terms of the Chern classes)

Example:

```
LIB "chern.lib";
ring r=0, (c(1..3)), ws(1,2,3);
list l=c(1..3);
// the total Chern class of the second exterior power of a vector bundle of rank 3
print(chWedge2LP(3, l));
↦ 1+2*c(1)+c(1)^2+c(2)+c(1)*c(2)-c(3)
```

D.5.2.34 todd

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: todd(l [, n]); l a list of polynomials, n integer

Return: list of polynomials

Purpose: computes [the first n] terms of the Todd class

Note: returns an empty list if l is empty

Example:

```
LIB "chern.lib";
// the terms of the Todd class up to degree 5
// in terms of the Chern classes c(1), c(2), c(3), c(4), c(5)
ring r=0, (c(1..5)), dp;
list l=c(1..5);
print( todd( l ) );
↦ [1]:
↦    1/2*c(1)
↦ [2]:
↦    1/12*c(1)^2+1/12*c(2)
↦ [3]:
↦    1/24*c(1)*c(2)
↦ [4]:
↦    -1/720*c(1)^4+1/180*c(1)^2*c(2)+1/240*c(2)^2+1/720*c(1)*c(3)-1/720*c(4\
   )
↦ [5]:
↦    -1/1440*c(1)^3*c(2)+1/480*c(1)*c(2)^2+1/1440*c(1)^2*c(3)-1/1440*c(1)*c\
   (4)
// in the same situation compute only first two terms
print( todd(l, 2) );
```

```
↦ [1]:
↦    1/2*c(1)
↦ [2]:
↦    1/12*c(1)^2+1/12*c(2)
// compute the first 5 terms corresponding to the Chern classes c(1), c(2)
l=c(1..2);
print( todd(l, 5) );
↦ [1]:
↦    1/2*c(1)
↦ [2]:
↦    1/12*c(1)^2+1/12*c(2)
↦ [3]:
↦    1/24*c(1)*c(2)
↦ [4]:
↦    -1/720*c(1)^4+1/180*c(1)^2*c(2)+1/240*c(2)^2
↦ [5]:
↦    -1/1440*c(1)^3*c(2)+1/480*c(1)*c(2)^2
```

D.5.2.35 toddE

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: toddE(l); l a list of polynomials

Return: polynomial

Purpose: computes the highest relevant term of the Todd class

Note: returns an empty list if l is empty,
very inefficient because the elimination is used, included for comparison with todd(c)

Example:

```
LIB "chern.lib";
// first  3 terms of the Todd class in terms of the Chern classes c(1), c(2), c(3)
ring r=0, (c(1..3)), dp;
list l;
//first term
l=c(1);
print( toddE( l ) );
↦ 1/2*c(1)
// second term
l=c(1..2);
print( toddE( l ) );
↦ 1/12*c(1)^2+1/12*c(2)
// third term
l=c(1..3);
print( toddE( l ) );
↦ 1/24*c(1)*c(2)
```

D.5.2.36 Bern

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: Bern(n); n non-negative integer

Return: list of numbers

Purpose: computes the list of (second) Bernoulli numbers from B(0) to B(n)

Note: needs a base ring to be defined, returns an empty list if n is negative, uses the Akiyama-Tanigawa algorithm

Example:

```
LIB "chern.lib";
// first 10 Bernoulli numbers: B(0), ..., B(9)
ring r=0,(t), dp;
print( Bern(9) );
↦ [1]:
↦ 1
↦ [2]:
↦ 1/2
↦ [3]:
↦ 1/6
↦ [4]:
↦ 0
↦ [5]:
↦ -1/30
↦ [6]:
↦ 0
↦ [7]:
↦ 1/42
↦ [8]:
↦ 0
↦ [9]:
↦ -1/30
↦ [10]:
↦ 0
```

D.5.2.37 tdCf

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: tdCf(n); n integer

Return: list of rational numbers

Purpose: computes up to degree n the coefficients of the Todd class of a line bundle

Note:

Example:

```
LIB "chern.lib";
// first 5 coefficients
ring r=0,(t), dp;
print( tdCf(4) );
↦ [1]:
↦ 1
↦ [2]:
↦ 1/2
↦ [3]:
↦ 1/12
↦ [4]:
↦ 0
↦ [5]:
↦ -1/720
```

D.5.2.38 tdTerms

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: tdTerms(n, f); n integer, f polynomial

Return: list of polynomials

Purpose: computes the terms of the Todd class of the line bundle with the Chern root f

Note:

Example:

```
LIB "chern.lib";
ring r=0, (t), ls;;
// the terms of the Todd class of a line bundle with Chern root t up to degree 4
print( tdTerms(4, t) );
↦ [1]:
↦    1
↦ [2]:
↦    1/2t
↦ [3]:
↦    1/12t2
↦ [4]:
↦    0
↦ [5]:
↦    -1/720t4
```

D.5.2.39 tdFactor

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: tdFactor(n, a); n integer, a polynomial

Return: polynomial

Purpose: computes up to degree n the Todd class
of the line bundle corresponding to the Chern root t

Note: returns 0 if n is negative

Example:

```
LIB "chern.lib";
// the Todd class up do degree 4
ring r=0,(t), ls;
print( tdFactor(4, t) );
↦ 1+1/2t+1/12t2-1/720t4
```

D.5.2.40 cProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: cProj(n); n integer

Return: list of integers

Purpose: computes the terms of positive degree of the total Chern class of the tangent bundle on the complex projective space

Note:

Example:

```
LIB "chern.lib";
ring r=0, (t), dp;
// the coefficients of the total Chern class of the complex projective line
print( cProj(1) );
↦ [1]:
↦    2
// the coefficients of the total Chern class of the complex projective plane
print( cProj(2) );
↦ [1]:
↦    3
↦ [2]:
↦    3
// the coefficients of the total Chern class of the complex projective space
// of dimension three
print( cProj(3) );
↦ [1]:
↦    4
↦ [2]:
↦    6
↦ [3]:
↦    4
```

D.5.2.41 chProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chProj(n); n integer

Return: list of (rational) numbers

Purpose: computes the terms of the Chern character of the tangent bundle on the complex projective space

Note:

Example:

```
LIB "chern.lib";
ring r=0, (t), dp;
// the coefficients of the Chern character of the complex projective line
print( chProj(1) );
↦ [1]:
↦ 1
↦ [2]:
↦ 2
// the coefficients of the Chern character of the complex projective plane
print( chProj(2) );
↦ [1]:
↦ 2
↦ [2]:
↦ 3
↦ [3]:
↦ 3/2
// the coefficients of the Chern character of the complex 3-dimensional projective sp
print( chProj(3) );
↦ [1]:
↦ 3
↦ [2]:
↦ 4
↦ [3]:
↦ 2
↦ [4]:
↦ 2/3
```

D.5.2.42 tdProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: tdProj(n); n integer

Return: list of (rational) numbers

Purpose: computes the terms of the Todd class
of the (tangent bundle of the) complex projective space

Note:

Example:

```
LIB "chern.lib";
ring r=0, (t), dp;
// the coefficients of the Todd class of the complex projective line
print( tdProj(1) );
↦ [1]:
↦    1
↦ [2]:
↦    1
// the coefficients of the Todd class of the complex projective line
print( tdProj(2) );
↦ [1]:
↦    1
↦ [2]:
↦    3/2
↦ [3]:
↦    1
// the coefficients of the Todd class of the complex projective line
print( tdProj(3) );
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    11/6
↦ [4]:
↦    1
```

D.5.2.43 eulerChProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: eulerChProj(n, r, c); n integer, r polynomial (or integer), c list of polynomials

Return: polynomial

Purpose: computes the Euler characteristic of a vector bundle on P_n in terms of its rank and Chern classes

Note:

Example:

```
LIB "chern.lib";
ring h=0, (r, c(1..3)),  ws(0,1,2,3);
list l=c(1..3);
// the Euler characteristic of a vector bundle on the projective line
print( eulerChProj(1, r, l) );
↦ r+c(1)
// the Euler characteristic of a vector bundle on the projective plane
print( eulerChProj(2, r, l) );
↦ r+3/2*c(1)+1/2*c(1)^2-c(2)
// the Euler characteristic of a vector bundle on P_3
print( eulerChProj(3, r, l) );
↦ r+11/6*c(1)+c(1)^2-2*c(2)+1/6*c(1)^3-1/2*c(1)*c(2)+1/2*c(3)
// assume now that we have a bundle framed at a subplane of P_3
// this implies c(1)=c(2)=0
l= 0, 0, c(3);
// the Euler characteristic is
print( eulerChProj(3, r, l) );
↦ r+1/2*c(3)
// which implies that c(3) must be even in this case
```

D.5.2.44 chNumbersProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chNumbersProj(n); n integer

Return: list of integers

Purpose: computes the Chern numbers of the projective space P_n

Note:

Example:

```
LIB "chern.lib";
ring h=0, (t), dp;
// The Chern numbers of the projective plane P_2:
print( chNumbersProj(2) );
↦ [1]:
↦    9
↦ [2]:
↦    3
// The Chern numbers of P_3:
print( chNumbersProj(3) );
↦ [1]:
```

```
↦    64
↦ [2]:
↦    24
↦ [3]:
↦    4
```

D.5.2.45 classpoly

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: classpoly(l, t); l list of polynomials, t polynomial

Return: polynomial

Purpose: computes the polynomial in t with coefficients being the entries of l

Note:

Example:

```
LIB "chern.lib";
ring r=0, (c(1..5), t), ds;
list l=c(1..5);
// get the polynomial c(1)*t + c(2)*t^2 + ... + c(5)*t^5
print( classpoly(l, t) );
↦ c(1)*t+c(2)*t^2+c(3)*t^3+c(4)*t^4+c(5)*t^5
```

D.5.2.46 chernPoly

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chernPoly(c, t); c list of polynomials, t polynomial

Return: polynomial

Purpose: computes the Chern polynomial in t

Note: does the same as toddPoly(...)

Example:

```
LIB "chern.lib";
ring r=0, (c(1..5), t), ds;
list l=c(1..5);
// get the Chern polynomial 1 + c(1)*t + c(2)*t^2 + ... + c(5)*t^5
print( chernPoly(l, t) );
↦ 1+c(1)*t+c(2)*t^2+c(3)*t^3+c(4)*t^4+c(5)*t^5
```

D.5.2.47 chernCharPoly

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: chernCharPoly(r, ch, t); r polynomial, ch list of polynomials, t polynomial

Return: polynomial

Purpose: computes the polynomial in t corresponding to the Chern character

Note:

Example:

```
LIB "chern.lib";
ring h=0, (r, ch(1..5), t), ds;
list l=ch(1..5);
// get the polynomial r + ch(1)*t + ch(2)*t^2 + ... + ch(5)*t^5
print( chernCharPoly(r, l, t) );
↦ r+ch(1)*t+ch(2)*t^2+ch(3)*t^3+ch(4)*t^4+ch(5)*t^5
```

D.5.2.48 toddPoly

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: toddPoly(td, t); td list of polynomials, t polynomial

Return: polynomial

Purpose: computes the polynomial in t corresponding to the Todd class

Note: does the same as chernPoly(...)

Example:

```
LIB "chern.lib";
ring r=0, (td(1..5), c(1..5), t), ds;
list l=td(1..5);
// get the polynomial 1 + td(1)*t + td(2)*t^2 + ... + td(5)*t^5
print( toddPoly(l, t) );
↦ 1+td(1)*t+td(2)*t^2+td(3)*t^3+td(4)*t^4+td(5)*t^5
```

D.5.2.49 rHRR

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: rHRR(N, ch, td); N integer, ch, td lists of polynomials

Return: polynomial

Purpose: computes the the main ingredient of the right-hand side of the Hirzebruch-Riemann-Roch formula

Note: in order to get the right-hand side of the HRR formula
one needs to be able to compute the degree of the output of this procedure

Example:

```
LIB "chern.lib";
ring r=0, (td(0..3), ch(0..3)), dp;
// Let ch(0), ch(1), ch(2), ch(3) be the terms of the Chern character
// of a vector bundle E on a 3-fold X.
list c = ch(0..3);
// Let td(0), td(1), td(2), td(3) be the terms of the Todd class of X.
list t = td(0..3);
// Then the highest term of the product ch(E).td(X) is:
print( rHRR(3, c, t) );
↦ td(3)*ch(0)+td(2)*ch(1)+td(1)*ch(2)+td(0)*ch(3)
```

D.5.2.50 SchurS

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: SchurS(I, S); I list of integers representing a partition, S list of polynomials

Return: poly

Purpose: computes the Schur polynomial in the Segre classes S (of the dual vector bundle), i.e., in the complete homogeneous symmetric polynomials, with respect to the partition I

Note: if S are the Segre classes of the tautological bundle on a grassmanian, this gives the cohomology class of a Schubert cycle

Example:

```
LIB "chern.lib";
// The Schur polynomial corresponding to the partition 1,2,4
// and the Segre classes 1, s(1), s(2),..., s(6)
ring r=0,(s(1..6)), dp;
list I=1,2,4;
list S=s(1..6);
print( SchurS(I, S) );
↦ s(1)*s(2)*s(4)-s(1)^2*s(5)-s(3)*s(4)+s(1)*s(6)
// compare this with the Schur polynomial computed using Chern classes
list C=chDual(chern(S));
print( SchurCh(I, C) );
↦ s(1)*s(2)*s(4)-s(1)^2*s(5)-s(3)*s(4)+s(1)*s(6)
```

D.5.2.51 SchurCh

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: SchurCh(I, C); I list of integers representing a partition, C list of polynomials

Return: poly

Purpose: computes the Schur polynomial in the Chern classes C, i.e., in the elementary symmetric polynomials, with respect to the partition I

Note: if C are the Chern classes of the tautological bundle on a grassmanian, this gives the cohomology class of a Schubert cycle

Example:

```
LIB "chern.lib";
// The Schur polynomial corresponding to the partition 1,2,4
// and the Chern classes c(1), c(2), c(3)
ring r=0,(c(1..3)), dp;
list I=1,2,4;
list C=c(1..3);
print( SchurCh(I, C) );
↦ c(1)^2*c(2)*c(3)-c(2)^2*c(3)-c(1)*c(3)^2
// Compare this with the Schur polynomial computed using Segre classes
list S=segre( chDual( list(c(1..3)) ), 6 );
print(SchurS(I,S));
↦ c(1)^2*c(2)*c(3)-c(2)^2*c(3)-c(1)*c(3)^2
```

D.5.2.52 part

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: part(m, n); m positive integer, n non-negative integer

Return: list of lists

Purpose: computes all partitions of integers not exceeding n into m non-negative summands

Note: if n is negative or m is non-positive, the list with one empty entry is returned

Example:

```
LIB "chern.lib";
// partitions into 3 summands of numbers not exceeding 1
print( part(3, 1) );
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦ [2]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       1
↦ [3]:
↦    [1]:
↦       0
↦    [2]:
↦       1
↦    [3]:
↦       1
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.5.2.53 dualPart

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: dualPart(I [,N]); I list of integers, N integer

Return: list of integers

Purpose: computes the partition dual (conjugate) to I

Note: the result is extended by zeroes to length N if an optional integer parameter N is given and the length of the computed dual partition is smaller than N

Example:

```
LIB "chern.lib";
// dual partition to (1, 3, 4):
list I = 1, 3, 4;
print( dualPart(I) );
↦ [1]:
```

```
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    2
↦ [4]:
↦    3
```

D.5.2.54 PartC

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: PartC(I, m); I list of integers, m integer

Return: list of integers

Purpose: computes the complement of a partition with respect to m

Note: returns the zero partition if the maximal element of the partition is smaller than m

Example:

```
LIB "chern.lib";
// Complement of the partition (1, 3, 4) with respect to 5
list I = 1, 3, 4;
print( PartC(I, 5) );
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    4
```

D.5.2.55 partOver

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: partOver(n, J); n integer, J list of integers (partition)

Return: list of lists

Purpose: computes the partitions over a given one with summands not exceeding n

Note:

Example:

```
LIB "chern.lib";
// Partitions over the partition (3, 3, 4) with summands not exceeding 4
list I = 3, 3, 4;
print( partOver(4, I) );
↦ [1]:
↦    [1]:
↦       3
↦    [2]:
↦       3
↦    [3]:
↦       4
↦ [2]:
```

```
↦     [1]:
↦        3
↦     [2]:
↦        4
↦     [3]:
↦        4
↦  [3]:
↦     [1]:
↦        4
↦     [2]:
↦        4
↦     [3]:
↦        4
```

D.5.2.56 partUnder

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: partUnder(J); J list of integers (partition)

Return: list of lists

Purpose: computes the partitions under a given one

Note:

Example:

```
LIB "chern.lib";
// Partitions under the partition (0, 1, 1)
list I = 0, 1, 1;
print( partUnder(I) );
↦  [1]:
↦     [1]:
↦        0
↦     [2]:
↦        0
↦     [3]:
↦        0
↦  [2]:
↦     [1]:
↦        0
↦     [2]:
↦        0
↦     [3]:
↦        1
↦  [3]:
↦     [1]:
↦        0
↦     [2]:
↦        1
↦     [3]:
↦        1
```

D.5.2.57 SegreA

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: SegreA(I); I an ideal

Return: list of integers

Purpose: computes the Segre classes of the subscheme defined by I

Note:

Example:

```
LIB "chern.lib";
// Consider a 3-dimensional projective space
ring r = 0, (x, y, z, w), dp;
// Consider 3 non-coplanar lines trough one point and compute the Segre class
ideal I=xy, xz, yz;
I;
↦ I[1]=xy
↦ I[2]=xz
↦ I[3]=yz
SegreA(I);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    -10
// Now consider 3 coplanar lines trough one point and its Segre class
ideal J=w, x*y*(x+y);
J;
↦ J[1]=w
↦ J[2]=x2y+xy2
SegreA(J);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    -12
```

D.5.2.58 FultonA

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: FultonA(I); I an ideal

Return: list of integers

Purpose: computes the Fulton classes of the subscheme defined by I

Note:

Example:

```
LIB "chern.lib";
// Consider a 3-dimensional projective space
```

```
ring r = 0, (x, y, z, w), dp;
// Consider 3 non-coplanar lines trough one point and compute the Fulton class
ideal I=xy, xz, yz;
I;
↦ I[1]=xy
↦ I[2]=xz
↦ I[3]=yz
FultonA(I);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    2
// Now consider 3 coplanar lines trough one point and its Fulton class
ideal J=w, x*y*(x+y);
J;
↦ J[1]=w
↦ J[2]=x2y+xy2
FultonA(J);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    0
```

D.5.2.59 CSMA

Procedure from library chern.lib (see Section D.5.2 [chern_lib], page 1390).

Usage: CSMA(I); I an ideal

Return: list of integers

Purpose: computes the Chern-Schwartz-MacPherson classes of the variety defined by I

Note:

Example:

```
LIB "chern.lib";
// consider the projective plane with homogeneous coordinates x, y, z
ring r = 0, (x, y, z), dp;
// the Chern-Schwartz-MacPherson class of a smooth cubic:
ideal I=x3+y3+z3;
I;
↦ I[1]=x3+y3+z3
CSMA(I);
↦ [1]:
↦    0
↦ [2]:
```

```
↦    3
↦ [3]:
↦    0
// the Chern-Schwartz-MacPherson class of singular cubic
// that is a union of 3 non-collinear lines:
ideal J=x*y*z;
J;
↦ J[1]=xyz
CSMA(J);
↦ [1]:
↦    0
↦ [2]:
↦    3
↦ [3]:
↦    3
// the Chern-Schwartz-MacPherson class of singular cubic
// that is a union of 3 lines passing through one point
ideal K=x*y*(x+y);
K;
↦ K[1]=x2y+xy2
CSMA(K);
↦ [1]:
↦    0
↦ [2]:
↦    3
↦ [3]:
↦    4
```

D.5.2.60 EulerAff

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: EulerAff(I); I an ideal

Return: integer

Purpose: computes the Euler characteristic of the affine variety defined by I

Note:

Example:

```
LIB "chern.lib";
ring r = 0, (x, y), dp;
// compute the Euler characteristic of the affine elliptic curve y^2=x^3+x+1;
ideal I=y2-x3-x-1;
EulerAff(I);
↦ -1
```

D.5.2.61 EulerProj

Procedure from library `chern.lib` (see Section D.5.2 [chern_lib], page 1390).

Usage: EulerProj(I); I an ideal

Return: integer

Purpose: computes the highest degree term of the Chern-Schwartz-MacPherson class of the variety defined by I, which equals the Euler characteristic

Note: uses CSMA(...)

Example:

```
LIB "chern.lib";
// consider the projective plane with homogeneous coordinates x, y, z
ring r = 0, (x, y, z), dp;
// Euler characteristic of a smooth cubic:
ideal I=x3+y3+z3;
I;
↦ I[1]=x3+y3+z3
EulerProj(I);
↦ 0
// Euler characteritic of 3 non-collinear lines:
ideal J=x*y*z;
J;
↦ J[1]=xyz
EulerProj(J);
↦ 3
// Euler characteristic of 3 lines passing through one point
ideal K=x*y*(x+y);
K;
↦ K[1]=x2y+xy2
EulerProj(K);
↦ 4
```

D.5.3 deRham_lib

Library: deRham.lib

Purpose: Computation of deRham cohomology

Authors: Cornelia Rottner, rottner@mathematik.uni-kl.de

Overview: A library for computing the de Rham cohomology of complements of complex affine varieties.

References:

[OT] Oaku, T.; Takayama, N.: Algorithms of D-modules - restriction, tensor product, localization, and local cohomology groups}, J. Pure Appl. Algebra 156, 267-308 (2001)
[R] Rottner, C.: Computing de Rham Cohomology,diploma thesis (2012)
[W1] Walther, U.: Algorithmic computation of local cohomology modules and the local cohomological dimension of algebraic varieties}, J. Pure Appl. Algebra 139, 303-321 (1999)
[W2] Walther, U.: Algorithmic computation of de Rham Cohomology of Complements of Complex Affine Varieties}, J. Symbolic Computation 29, 796-839 (2000)
[W3] Walther, U.: Computing the cup product structure for complements of complex affine varieties, J. Pure Appl. Algebra 164, 247-273 (2001)

Procedures:

D.5.3.1 deRhamCohomology

Procedure from library `deRham.lib` (see Section D.5.3 [deRham_lib], page 1426).

Usage: deRhamCohomology(L[,choices]); L a list consisting of polynomials, choices optional list consisting of one up to three strings
The optional strings may be one of the strings
-'noCE': compute quasi-isomorphic complexes without using Cartan-Eilenberg resolutionsq
-'Vdres': compute quasi-isomorphic complexes using Cartan-Eilenberg resolutions; the CE resolutions are computed via V__d-homogenization and without using Schreyer's method
-'Sres': compute quasi-isomorphic complexes using Cartan-Eilenberg resolutions in the homogenized Weyl algebra via Schreyer's method
one of the strings
-'iterativeloc': compute localizations by factorizing the polynomials and successive localization of the factors
-'no iterativeloc': compute localizations by directly localizing the product
and one of the strings
-'onlybounds': computes bounds for the minimal and maximal integer roots of the global b-function
-'exactroots' computes the minimal and maximal integer root of the global b-function
The default is 'noCE', 'iterativeloc' and 'onlybounds'.

Assume: -The basering must be a polynomial ring over the field of rational numbers

Return: list, where the ith entry is the (i-1)st de Rham cohomology group of the complement of the complex affine variety given by the polynomials in L

Example:

```
LIB "deRham.lib";
ring r = 0,(x,y,z),dp;
list L=(xy,xz);
deRhamCohomology(L);
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    0
↦ [4]:
↦    1
↦ [5]:
↦    1
```

D.5.3.2 MVComplex

Procedure from library `deRham.lib` (see Section D.5.3 [deRham_lib], page 1426).

Usage: MVComplex(L); L a list of polynomials

Assume: -Basering is a polynomial ring with n vwariables and rational coefficients -L is a list of non-constant polynomials

Return: ring W: the nth Weyl algebra
W contains a list MV, which represents the Mayer-Vietrois complex (C^i,d^i) of the polynomials contained in L as follows:

the C^i are given by D_n^ncols(C[2*i-1])/im(C[2*i-1]) and the differentials d^i are given by C[2*i]

Example:

```
LIB "deRham.lib";
ring r = 0,(x,y,z),dp;
list L=xy,xz;
def C=MVComplex(L);
setring C;
MV;
↦ [1]:
↦    _[1,1]=D(3)
↦    _[1,2]=0
↦    _[2,1]=x(1)*D(1)+1
↦    _[2,2]=0
↦    _[3,1]=-x(2)*D(2)-1
↦    _[3,2]=0
↦    _[4,1]=0
↦    _[4,2]=D(2)
↦    _[5,1]=0
↦    _[5,2]=x(1)*D(1)+1
↦    _[6,1]=0
↦    _[6,2]=-x(3)*D(3)-1
↦ [2]:
↦    _[1,1]=-x(1)*x(3)
↦    _[2,1]=x(1)*x(2)
↦ [3]:
↦    _[1,1]=x(2)*D(2)+1
↦    _[2,1]=x(1)*D(1)+2
↦    _[3,1]=-x(3)*D(3)-1
↦ [4]:
↦    _[1,1]=0
```

D.5.4 divisors_lib

Library: divisors.lib

Purpose: Divisors and P-Divisors

Authors: Janko Boehm boehm@mathematik.uni-kl.de
Lars Kastner kastner@math.fu-berlin.de
Benjamin Lorenz blorenz@math.uni-frankfurt.de
Hans Schoenemann hannes@mathematik.uni-kl.de
Yue Ren ren@mathematik.uni-kl.de

Overview: We implement a class divisor on an algebraic variety and methods for computing with them. Divisors are represented by tuples of ideals defining the positive and the negative part. In particular, we implement the group structure on divisors, computing global sections and testing linear equivalence.

In addition to this we provide a class formaldivisor which implements integer formal sums of divisors (not necessarily prime). A formal divisor can be evaluated to a divisor, and a divisor can be decomposed into a formal sum.

Finally we provide a class pdivisor which implements polyhedral formal sums of divisors (P-divisors) where the coefficients are assumed to be polyhedra with fixed tail cone.

There is a function to evaluate a P-divisor on a vector in the dual of the tail cone. The result will be a formal divisor.

References:
For the class divisor we closely follow Macaulay2's tutorial on divisors.

Procedures:

D.5.4.1 makeDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: makeDivisor(I ,J); I = ideal, J = ideal.

Assume: I and J are ideals in a qring Q of a smooth irreducible variety X such that any ideal in Q satisfies the S2 condition.

Return: a divisor on X

Theory: The procedure will eliminate all components which are not of codimension 1. The S2 condition requires that every proper nonzero principal ideal has pure codimension 1.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor P = makeDivisor(ideal(x,z),ideal(1));
```

D.5.4.2 divisorplus

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: divisorplus(A ,B); A + B; A = divisor, B = divisor.

Assume: A and B are divisors on X.

Return: a divisor on X

Theory: The procedure will compute the product of the numerator and denominator ideals, respectively.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
A+B;
↦ (x,yz) - (1)
↦
```

D.5.4.3 multdivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: multdivisor(n ,A); A*n; n = integer, A = divisor.

Assume: n is an integer and A is a divisor on X.

Return: a divisor on X

Theory: The procedure will compute the n-th power of the numerator and denominator ideals, respectively.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
A;
↦ (z,x) - (1)
↦
divisor D = multdivisor(4,A);
D;
↦ (z2,xz) - (1)
↦
A*4;
↦ (z2,xz) - (1)
↦
```

D.5.4.4 negativedivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: negativedivisor(A); A*(-1); A = divisor.

Assume: A is a divisor on X.

Return: a divisor on X

Theory: The procedure will interchange the numerator and denominator ideals.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
A;
↦ (z,x) - (1)
↦
divisor D = negativedivisor(A);
D;
↦ (1) - (z,x)
↦
```

D.5.4.5 normalForm

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: normalForm(A); A = divisor.

Assume: A is a divisor on X.

Return: different representative of the same divisor on X

Theory: The procedure will cancel common components of numerator and denominator.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
divisor D = (A+B)+multdivisor(-1,B);
D;
↦ (x,yz) - (y,x)
↦
normalForm(D);
↦ (z,x) - (1)
↦
```

D.5.4.6 isEqualDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: isEqualDivisor(A,B); A = divisor, B = divisor.

Assume: A and B are divisors on X.

Return: int 0 or 1, checks equality of A and B.

Theory: The procedure will compute the normal forms of A and B and compare.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
divisor D = (A+B)+multdivisor(-1,B);
isEqualDivisor(A,D);
↦ 1
```

D.5.4.7 globalSections

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: globalSections(A); A = divisor.

Assume: A is a divisor on X.

Return: a list with a basis of the space of global sections of D.

Theory: We assume that the qring of X satisfies the S2-condition and that X is smooth. We compute sat((f*J) : I) /f where D = (I)-(J).

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor P = makeDivisor(ideal(x,z),ideal(1));
divisor D = multdivisor(4,P);
globalSections(D);
↦ [1]:
↦    _[1]=x2
↦    _[2]=z2
↦    _[3]=yz
↦    _[4]=xz
↦ [2]:
↦    z2
```

D.5.4.8 degreeDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: degreeDivisor(A); A = divisor.

Assume: A is a divisor on X.

Return: The degree of A.

Theory: We compute difference of the degrees of the numerator and denominator ideals.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor P = makeDivisor(ideal(x,z),ideal(1));
degreeDivisor(P);
↦ 1
```

D.5.4.9 linearlyEquivalent

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: linearlyEquivalent(A,B); A = divisor, B = divisor.

Assume: A and B are divisors on X.

Return: list if A and B a linearly equivalent and int 0 otherwise.

Theory: Checks whether A-B is principle. If yes, returns a list L=(f,g) where A - B = (f/g).

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
linearlyEquivalent(A,B);
↦ 0
linearlyEquivalent(multdivisor(2,A),multdivisor(2,B));
↦ [1]:
↦    x
↦ [2]:
↦    z
```

D.5.4.10 effective

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: effective(A); A = divisor.

Assume: A is a divisor on X which is linearly equivalent to an effective divisor.

Return: divisor on X.

Theory: We compute an effective divisor linearly equivalent to A.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
divisor D = divisorplus(multdivisor(2,B),negativedivisor(A));
effective(D);
↦ (z,x) - (1)
↦
```

D.5.4.11 makeFormalDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: makeFormalDivisor(L); L = list.

Assume: L is a list of tuples of an integer and a divisor.

Return: a formal divisor on X

Theory: Represents an integer formal sum of divisors.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
```

```
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
makeFormalDivisor(list(list(-5,A),list(2,B)));
↦ -5*( (z,x) - (1) )
↦ +2*( (y,x) - (1) )
↦
```

D.5.4.12 evaluateFormalDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: evaluateFormalDivisor(D); D = formal divisor.

Assume: D is a formal divisor on X.

Return: a divisor on X

Theory: Will evaluate the formal sum.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
formaldivisor fE= makeFormalDivisor(list(list(-5,A),list(2,B)));
evaluateFormalDivisor(fE);
↦ (x,y2) - (z2,x2z)
↦
```

D.5.4.13 formaldivisorplus

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: formaldivisorplus(A ,B); A + B; A = formaldivisor, B = formaldivisor.

Assume: A and B are formal divisors on X.

Return: a formal divisor on X

Theory: The procedure will add the formal sums.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
divisor C = makeDivisor(ideal(x-z,y),ideal(1));
formaldivisor fE= makeFormalDivisor(list(list(-5,A),list(2,B)));
formaldivisor fE2= makeFormalDivisor(list(list(-5,A),list(2,C)));
formaldivisorplus(fE,fE2);
↦ [1]:
```

```
↦    [1]:
↦       -10
↦    [2]:
↦       (z,x) - (1)
↦
↦ [2]:
↦    [1]:
↦       2
↦    [2]:
↦       (y,x) - (1)
↦
↦ [3]:
↦    [1]:
↦       2
↦    [2]:
↦       (y,x-z) - (1)
↦
```

D.5.4.14 negativeformaldivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: negativeformaldivisor(A); A = formaldivisor.

Assume: A is a formaldivisor on X.

Return: a formal divisor on X

Theory: The procedure will change the signs of the coefficients.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
formaldivisor fE= makeFormalDivisor(list(list(-5,A),list(2,B)));
negativeformaldivisor(fE);
↦ 5*( (z,x) - (1) )
↦ -2*( (y,x) - (1) )
↦
```

D.5.4.15 multformaldivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: multformaldivisor(n ,A); A*n; n = integer, A = formaldivisor.

Assume: n is an integer and A is a formal divisor on X.

Return: a formal divisor on X

Theory: The procedure will multiply the formal sum with n.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
formaldivisor fE= makeFormalDivisor(list(list(-5,A),list(2,B)));
fE*2;
↦ -10*( (z,x) - (1) )
↦ +4*( (y,x) - (1) )
↦
```

D.5.4.16 degreeFormalDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: degreeFormalDivisor(A); A = formaldivisor.

Assume: A is a formaldivisor on X.

Return: The degree of A.

Theory: We compute degrees of the summands and return the weighted sum.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
formaldivisor fE= makeFormalDivisor(list(list(-5,A),list(2,B)));
degreeFormalDivisor(fE);
↦ -3
```

D.5.4.17 makePDivisor

Procedure from library `divisors.lib` (see Section D.5.4 [divisors_lib], page 1428).

Usage: makePDivisor(L); L = list.

Assume: L is a list of tuples of a integral polyhedron and a divisor such that all polyhedra have the same tail cone.

Return: a pdivisor on X

Theory: Represents an polyhedral formal sum of divisors.

Example:

```
LIB "divisors.lib";
ring r=31991,(x,y,z),dp;
ideal I = y^2*z - x*(x-z)*(x+3*z);
qring Q = std(I);
divisor A = makeDivisor(ideal(x,z),ideal(1));
divisor B = makeDivisor(ideal(x,y),ideal(1));
```

```
intmat M[4][4]= 1,4,0,0,
1,0,3,0,
0,0,0,2,
1,1,1,1;
polytope PP = polytopeViaPoints(M);
makePDivisor(list(list(PP,A),list(PP,B)));
↦ tail=<cone>
↦ summands=<list>
```

D.5.5 goettsche_lib

Library: goettsche.lib

Purpose: Drezet's formula for the Betti numbers of the moduli space of Kronecker modules; Goettsche's formula for the Betti numbers of the Hilbert scheme of points on a surface; Nakajima's and Yoshioka's formula for the Betti numbers of the punctual Quot-schemes on a plane or, equivalently, of the moduli spaces of the framed torsion-free planar sheaves; Macdonald's formula for the symmetric product

Author: Oleksandr Iena, o.g.yena@gmail.com

References:

[1] Drezet, Jean-Marc Cohomologie des varie'te's de modules de hauter nulle. Mathematische Annalen: 281, 43-85, (1988).

[2] Goettsche, Lothar, The Betti numbers of the Hilbert scheme of points on a smooth projective surface.
Mathematische Annalen: 286, 193-208, (1990).

[3] Macdonald, I. G., The Poincare polynomial of a symmetric product, Mathematical proceedings of the Cambridge Philosophical Society: 58, 563-568, (1962).

[4] Nakajima, Hiraku; Lectures on instanton counting, CRM Proceedings and Lecture Notes, Yoshioka, Kota Volume 88, 31-101, (2004).

Procedures:

D.5.5.1 GoettscheF

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: GoettscheF(z, t, n, b); z, t polynomials, n integer, b list of non-negative integers

Return: polynomial in z and t

Purpose: computes the Goettsche's formula up to degree n in t

Note: zero is returned if n<0 or b is not a list of non-negative integers or if there are not enough Betti numbers

Example:

```
LIB "goettsche.lib";
ring r=0, (t, z), ls;
// consider the projective plane with Betti numbers 1,0,1,0,1
list b=1,0,1,0,1;
// get the Goettsche's formula up to degree 3
print( GoettscheF(z, t, 3, b) );
↦ 1+t+tz2+tz4+t2+2t2z2+3t2z4+2t2z6+t2z8+t3+2t3z2+5t3z4+6t3z6+5t3z8+2t3z10+t\
   3z12
```

D.5.5.2 PPolyH

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: PPolyH(z, n, b); z polynomial, n integer, b list of non-negative integers

Return: polynomial in z

Purpose: computes the Poincare polynomial of the Hilbert scheme of n points on a surface with Betti numbers b

Note: zero is returned if n<0 or b is not a list of non-negative integers or if there are not enough Betti numbers

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// consider the projective plane P_2 with Betti numbers 1,0,1,0,1
list b=1,0,1,0,1;
// get the Poincare polynomial of the Hilbert scheme of 3 points on P_2
print( PPolyH(z, 3, b) );
↦ 1+2z2+5z4+6z6+5z8+2z10+z12
```

D.5.5.3 BettiNumsH

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: BettiNumsH(n, b); n integer, b list of non-negative integers

Return: list of non-negative integers

Purpose: computes the Betti numbers of the Hilbert scheme
of n points on a surface with Betti numbers b

Note: an empty list is returned if n<0 or b is not a list of non-negative integers or if there are not enough Betti numbers

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// consider the projective plane P_2 with Betti numbers 1,0,1,0,1
list b=1,0,1,0,1;
// get the Betti numbers of the Hilbert scheme of 3 points on P_2
print( BettiNumsH(3, b) );
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    2
↦ [4]:
↦    0
↦ [5]:
↦    5
↦ [6]:
↦    0
↦ [7]:
```

```
↦     6
↦  [8]:
↦     0
↦  [9]:
↦     5
↦  [10]:
↦     0
↦  [11]:
↦     2
↦  [12]:
↦     0
↦  [13]:
↦     1
```

D.5.5.4 NakYoshF

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: NakYoshF(z, t, r, n); z, t polynomials, r, n integers

Return: polynomial in z and t

Purpose: computes the formula of Nakajima and Yoshioka
up to degree n in t

Note: zero is returned if n<0 or r<=0

Example:

```
LIB "goettsche.lib";
ring r=0, (t, z), ls;
// get the Nakajima-Yoshioka formula for r=1 up to degree 3, i.e.,
// the generating function for the Poincare polynomials of the
// punctual Hilbert schemes of n planar points
print( NakYoshF(z, t, 1, 3) );
↦ 1+t+t2+t2z2+t3+t3z2+t3z4
```

D.5.5.5 PPolyQp

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: PPolyQp(z, r, n); z polynomial, r, n integers

Return: polynomial in z

Purpose: computes the Poincare polynomial of the punctual Quot-scheme of rank r on n planar points

Note: zero is returned if n<0 or r<=0

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// get the Poincare polynomial of the punctual Hilbert scheme (r=1)
// of 3 planar points
print( PPolyQp(z, 1, 3) );
↦ 1+z2+z4
```

D.5.5.6 BettiNumsQp

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: BettiNumsQp(r, n); n, r integers

Return: list of non-negative integers

Purpose: computes the Betti numbers of the punctual Quot-scheme of rank r on n points on a plane

Note: an empty list is returned if n<0 or r<=0

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// get the Betti numbers of the punctual Hilbert scheme (r=1)
// of 3 points on a plane
print( BettiNumsQp(1, 3) );
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    1
↦ [4]:
↦    0
↦ [5]:
↦    1
```

D.5.5.7 MacdonaldF

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: MacdonaldF(z, t, n, b); z, t polynomials, n integer, b list of non-negative integers

Return: polynomial in z and t with integer coefficients

Purpose: computes the Macdonalds's formula up to degree n in t

Note: zero is returned if n<0 or b is not a list of non-negative integers

Example:

```
LIB "goettsche.lib";
ring r=0, (t, z), ls;
// consider the projective plane with Betti numbers 1,0,1,0,1
list b=1,0,1,0,1;
// get the Macdonald's formula up to degree 3
print( MacdonaldF(z, t, 3, b) );
↦ 1+t+tz2+tz4+t2+t2z2+2t2z4+t2z6+t2z8+t3+t3z2+2t3z4+2t3z6+2t3z8+t3z10+t3z12
```

D.5.5.8 PPolyS

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: PPolyS(z, n, b); z polynomial, n integer, b list of non-negative integers

Return: polynomial in z with integer coefficients

Purpose: computes the Poincare polynomial of the n-th symmetric power of a variety with Betti numbers b

Note: zero is returned if n<0 or b is not a list of non-negative integers

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// consider the projective plane P_2 with Betti numbers 1,0,1,0,1
list b=1,0,1,0,1;
// get the Poincare polynomial of the third symmetric power of P_2
print( PPolyS(z, 3, b) );
↦ 1+z2+2z4+2z6+2z8+z10+z12
```

D.5.5.9 BettiNumsS

Procedure from library goettsche.lib (see Section D.5.5 [goettsche_lib], page 1437).

Usage: BettiNumsS(n, b); n integer, b list of non-negative integers

Return: list of non-negative integers

Purpose: computes the Betti numbers of the n-th symmetric power of a variety with Betti numbers b

Note: an empty list is returned if n<0 or b is not a list of non-negative integers

Example:

```
LIB "goettsche.lib";
ring r=0, (z), ls;
// consider a complex torus T (elliptic curve) with Betti numbers 1,2,1
list b=1,2,1;
// get the Betti numbers of the second symmetric power of T
print( BettiNumsS(2, b) );
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    2
↦ [4]:
↦    2
↦ [5]:
↦    1
// consider a projective plane P_2 with Betti numbers 1,0,1,0,1
b=1,0,1,0,1;
// get the Betti numbers of the third symmetric power of P_2
print( BettiNumsS(3, b) );
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    1
↦ [4]:
↦    0
↦ [5]:
↦    2
↦ [6]:
↦    0
↦ [7]:
↦    2
↦ [8]:
↦    0
↦ [9]:
↦    2
↦ [10]:
↦    0
↦ [11]:
↦    1
↦ [12]:
↦    0
↦ [13]:
↦    1
```

D.5.5.10 PPolyN

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: PPolyN(t, q, m, n); t polynomial, q, m, n integers

Return: polynomial in t

Purpose: computes the Poincare polynomial of the moduli space of Kronecker modules N(q; m, n)

Note: if m and n are not coprime, the result does not necessary make sense

Example:

```
LIB "goettsche.lib";
ring r=0, (t), ls;
// get the Poincare polynomial of N(3; 2, 3)
print( PPolyN(t, 3, 2, 3) );
↦ 1+t2+3t4+3t6+3t8+t10+t12
```

D.5.5.11 BettiNumsN

Procedure from library `goettsche.lib` (see Section D.5.5 [goettsche_lib], page 1437).

Usage: BettiNumsN(q, m, n); q, m, n integers

Return: list of integers

Purpose: computes the Betti numbers of the moduli space
of Kronecker modules N(q; m, n)

Note: if m and n are not coprime, the result does not necessary make sense

Example:

```
LIB "goettsche.lib";
ring r=0, (t), dp;
// get the Betti numbers of N(3; 2, 3)
print( BettiNumsN(3, 2, 3) );
```

```
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    1
↦ [4]:
↦    0
↦ [5]:
↦    3
↦ [6]:
↦    0
↦ [7]:
↦    3
↦ [8]:
↦    0
↦ [9]:
↦    3
↦ [10]:
↦    0
↦ [11]:
↦    1
↦ [12]:
↦    0
↦ [13]:
↦    1
```

D.5.6 graal_lib

Library: graal.lib

Purpose: localization at prime ideals and their associated graded rings

Author: Magdaleen Marais, magdaleen@aims.ac.za
Yue Ren, ren@mathematik.uni-kl.de

Overview: This library is on a computational treatment of localizations at prime ideals and their associated graded rings based on a work of Mora. Not only does it construct a ring isomorphic to the localization of an affine coordinate ring at a prime ideal,
the algorithms in this library aim to exploit the topology in the localization by computing first and foremost in the associated graded ring and lifting the result to the localization afterwards.
Features include a check for regularity and the resolution of ideals.

References:

Mora, Teo: La queste del Saint Gr_a(A_L): A computational approach to local algebra
Marais, Magdaleen and Ren, Yue: Mora's holy graal: Algorithms for computing in localizations at prime ideals

Procedures:

D.5.6.1 graalMixed

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: graalMixed(L,t); L ideal, t int (optional)

Return: graalBearer with all the necessary structures for our machinery if t specified and t>0, puts an upper time limit
on finding a necessary transformation to map an intermediate ideal into general position.

Note: assumes that the current basering is a domain and that L is a prime ideal.

Example:

```
LIB "graal.lib";
// see [Mora] Example 6.5
ring Q = 0,(x,y,z),dp;
ideal H = y2-xz;
qring A = std(H);
ideal L = x3-yz,x2y-z2;
graalBearer Gr = graalMixed(L); Gr;
↦ affine coordinate ring:
↦    (QQ),(x,y,z),(dp(3),C)
↦      mod <y2-xz>
↦
↦ ideal defining the subvariety:
↦    <x3-yz,x2y-z2>
↦
↦ Al:
↦    (0,z),(Y(1),Y(2),x,y),(ds(2),c,dp(2))
↦      mod <x^3+(-z)*y-Y(1),x^2*y+(-z^2)-Y(2),y^2+(-z)*x,(z)*Y(1)-Y(2)*y>
↦ graal:
↦    (0,z),(Y(1),Y(2),y),(c,dp(2),dp(1))
↦      mod <(z)*Y(1)-Y(2)*y,y^5+(-z^4)>
↦    where
↦      Y(1) represents generator x3-yz
↦      Y(2) represents generator x2y-z2
↦    and x,y in Al are mapped to 1/(z)*y^2,y in Graal
↦
```

D.5.6.2 dimensionOfLocalization

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: dimensionOfLocalization(L); L ideal or graalBearer

Return: int, the dimension of the localization A_L of A at L.

Example:

```
LIB "graal.lib";
ring Q = 0,(X(1),X(2)),dp;
ideal H = X(2)^2-(X(1)-1)*X(1)*(X(1)+1);
ideal J = std(X(1),X(2));
↦ // ** _ is no standard basis
qring A = std(H);
ideal L = fetch(Q,J);
graalBearer Gr = graalMixed(L);
// def fA = Gr.fA; setring fA;
dimensionOfLocalization(Gr); // = 1
↦ // ** redefining scriptIin (    ideal scriptIin = Gr.scriptIin;) graal.li\
   b::dimensionOfLocalization:582
↦ 1
```

D.5.6.3 systemOfParametersOfLocalization

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: systemOfParametersOfLocalization(def L); L ideal or graalBearer

Return: ideal, a system of parameter of the localization A_L of A at L.

Example:

```
LIB "graal.lib";
ring Q = 0,(X(1),X(2)),dp;
ideal H = X(2)^2-(X(1)-1)*X(1)*(X(1)+1);
ideal J = X(1),X(2);
qring A = std(H);
ideal L = fetch(Q,J);
graalBearer Gr = graalMixed(L);
systemOfParametersOfLocalization(Gr); // = 1
↦ // ** redefining J (    ideal J = Gr.J; ideal ret;) graal.lib::systemOfPa\
   rametersOfLocalization:647
↦ _[1]=X(2)
```

D.5.6.4 isLocalizationRegular

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: isLocalizationRegular(def L); L ideal or graalBearer

Return: int, 1 if the localization A_L of A at L is regular,
0 otherwise.

Example:

```
LIB "graal.lib";
ring Q = 0,(X(1),X(2)),dp;
ideal H = X(2)^2-(X(1)-1)*X(1)*(X(1)+1);
ideal J = X(1),X(2);
qring A = std(H);
ideal L = fetch(Q,J);
graalBearer Gr = graalMixed(L);
isLocalizationRegular(Gr); // = 1
↦ 1
```

D.5.6.5 warkedPreimageStd

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: warkedPreimageStd(wM); M warkedModule

Return: given wM consisting of:
- wM.Gr a graalBearer containing all relevant global structures - wM.modQ0y generating set G of a module M over Q0y - wM.stdmodQ0y empty
- wM.qQ0y empty
- wM.modKy corresponding generating set H of M_in over Ky - wM.stdmodKy empty
- wM.qKy empty
- wM.w weights on M
returns the same warkedModule, except following differences: - wM.stdmodQ0y contains a subset G such that for any standard basis L of the kernel G + L is a standard

basis of modQ0y + kernel
- wM.qQ0y contains a transformation matrix such that stdmodAy = QAy*modQ0y
- wM.stdmodKy contains a standardbasis of modKy
- wM.qKy contains a transformation matrix such that stdmodKy = QKy*modKy

Note: the standard basis of modAy is computed by lifting a corresponding Groebner basis of modKy

Example:

```
LIB "graal.lib";
ring Q = 0,(x,y,z),dp;
ideal H = y2-xz;
qring A = std(H);
ideal L = x3-yz,x2y-z2;
graalBearer Gr = graalMixed(L);
def Q0y = Gr.Q0y; setring Q0y;
module M = (Y(1)*y+y^2-1)*gen(1)+(Y(2)*z+z^2-1)*gen(2), Y(1)*y*gen(1)+Y(2)*z*gen(2);
/* This is M: */
print(matrix(M));
↦ y^2-1+Y(1)*y,    Y(1)*y,
↦ (z^2-1)+(z)*Y(2),(z)*Y(2)
intvec w = 1,1,1;
warkedModule wM;
wM.Gr = Gr;
wM.modQ0y = M;
wM.w = w;
def Ky = Gr.Ky; setring Ky;
module Min = (y^2-1)*gen(1)+(z^2-1)*gen(2),Y(1)*y*gen(1)+Y(2)*z*gen(2);
/* This is M_in: */
print(matrix(Min));
↦ y^2-1,  Y(1)*y,
↦ (z^2-1),(z)*Y(2)
wM.modKy = Min;
/* warkedPreimageStd yields the same standard basis as std: */
warkedModule wN = warkedPreimageStd(wM); wN;
↦ // ** redefining scriptIin (  ideal scriptIin = Gr.scriptIin;) graal.lib:\
  :warkedPreimageStd:892
↦ // ** redefining scriptI (  ideal scriptI = Gr.scriptI;) graal.lib::warke\
  dPreimageStd:921
↦ // ** redefining a (  poly a = L[1][1,1];) graal.lib::normalizeInY:810
↦ module over Q^0[Y] = (0,z),(Y(1),Y(2),x,y),(ds(2),c,dp(2)) / <0>:
↦ y^2-1+Y(1)*y,    Y(1)*y,
↦ (z^2-1)+(z)*Y(2),(z)*Y(2)
↦ standard basis:
↦ _[1,1],_[1,2],0,              0,                0,         0,
↦ _[2,1],_[2,2],x^3+(-z)*y-Y(1),x^2*y+(-z^2)-Y(2),y^2+(-z)*x,(z)*Y(1)-Y(2)*\
  y
↦ module over K[Y] = (0,z),(Y(1),Y(2),y),(c,dp(2),dp(1)) / <y^5+(-z^4)>:
↦ y^2-1,  Y(1)*y,
↦ (z^2-1),(z)*Y(2)
↦ weights on the unit vectors: 1,1,1
↦ 
setring Q0y;
module stdM = std(M);
```

```
print(matrix(stdM));
↦ y^2-1+Y(1)*y,    Y(1)-Y(1)^2*y,Y(1)^2*y^2,
↦ (z^2-1)+(z)*Y(2),_[2,2],       _[2,3]
```

D.5.6.6 resolutionInLocalization

Procedure from library `graal.lib` (see Section D.5.6 [graal_lib], page 1443).

Usage: resolutionInLocalization(I,L); I ideal, L ideal or graalBearer

Return: the resolution of I*A_L, where
A_L is the localization of the current basering (possibly a quotient ring) at a prime ideal L.

Example:

```
LIB "graal.lib";
ring Q = 0,(x,y,z,w),dp;
ideal circle = (x-1)^2+y^2-3,z;
ideal twistedCubic = xz-y2,yw-z2,xw-yz,z;
ideal I = std(intersect(circle,twistedCubic));
// the resolution is more complicated due to the twisted cubic
res(I,0);
↦  1      4      5      2
↦ Q <--  Q <--  Q <--  Q
↦ 
↦ 0      1      2      3
↦ resolution not minimized yet
↦ 
// however if we localize outside of the twisted cubic,
// it should become very easy again.
ideal L = std(I+ideal(x-1));
graalBearer Gr = graalMixed(L); Gr;
↦ affine coordinate ring:
↦    (QQ),(x,y,z,w),(dp(4),C)
↦ 
↦ ideal defining the subvariety:
↦    <z,x-1,y2w-3w,y4-3y2>
↦ 
↦ Al:
↦    (0,w),(Y(1),Y(2),Y(3),Y(4),x,y,z),(ds(4),c,dp(3))
↦      mod <(w)*y^2+(-3*w)-Y(3),x-1-Y(2),z-Y(1),(3*w)*Y(3)+(-w^2)*Y(4)+Y(3)\
^2>
↦ graal:
↦    (0,w),(Y(1),Y(2),Y(3),Y(4),z),(c,dp(4),dp(1))
↦      mod <3*Y(3)+(-w)*Y(4),z^2+10*z-2>
↦    where
↦      Y(1) represents generator z
↦      Y(2) represents generator x-1
↦      Y(3) represents generator y2w-3w
↦      Y(4) represents generator y4-3y2
↦    and x,y,z in Al are mapped to 1,1/3*z+5/3,0 in Graal
↦ 
markedResolution mr = resolutionInLocalization(I,Gr);
↦ // ** full resolution in a qring may be infinite, setting max length to 5
```

```
mr;
↦ resolution over Al:
↦   1       2       1
↦ Al <--  Al <--  Al
↦
↦ 0       1       2
↦ resolution not minimized yet
↦
↦ k=1
↦ Y(1),(w^2)*Y(4)+(3*w^2)*Y(2)^2*x+(3*w)*Y(2)*Y(3)-Y(3)^2
↦
↦ k=2
↦ _[1,1],
↦ -Y(1)
↦
↦ resolution over Graal:
↦      1          2          1
↦ Graal <--  Graal <--  Graal
↦
↦ 0          1          2
↦
↦ k=1
↦ Y(1),(w^2)*Y(4)
↦
↦ k=2
↦ (w^2)*Y(4),
↦ -Y(1)
↦
↦
```

D.5.7 hess_lib

Library: hess.lib

Purpose: Riemann-Roch space of divisors on function fields and curves

Authors: I. Stenger: stenger@mathematik.uni-kl.de

Overview: Let f be an absolutely irreducible polynomial in two variables x,y. Assume that f is monic as a polynomial in y. Let F = Quot(k[x,y]/f) be the function field defined by f. Define O_F = IntCl(k[x],F) and O_(F,inf) = IntCl(k[1/x],F). We represent a divisor D on F by two fractional ideals I and J of O_F and O_(F,inf), respectively. The Riemann-Roch space L(D) is then the intersection of I^(-1) and J^(-1).

Procedures:

D.5.7.1 RiemannRochHess

Procedure from library `hess.lib` (see Section D.5.7 [hess_lib], page 1448).

Note: All fractional ideals must represented by a list of size two. The first element is an ideal of k[x,y] and the second element the common denominator, i.e, a polynomial of k[x].

Assume: The base ring R must be a ring in two variables, say x,y, or three variables, say x,y,z. If nvars(R) = 2:

- f is an absolutely irreducible polynomial, monic as a polynomial in y.
- List divisorD describes a divisor D of F = Quot(k[x,y]/f). If (s = "ideals") D is given in ideal representation, i.e., divisorD is a list of size 2.
divisorD[1] is the finite ideal of D, i.e., the
fractional ideal of D of IntCl(k[x],F).
divisorD[2] is the infinite ideal of D, i.e,
the fractional ideal of D of IntCl(k[1/x],F).
If (s = "free")
D is given in free representation, i.e., divisorD is a list of size 2, containing the finite and infinite places of D with exponents.
divisorD[i], i = 1,2, is a list. Each element of the list is again a list. The first entry is a fractional ideal, and the second an integer, the exponent of the place. If nvars(R) = 3:
- f is an absolutely irreducible homogeneous polynomial describing the projective plane curve corresponding to the function field F. We assume that the dehomogenization of f w.r.t. z is monic as a polynomial in y.
List divisorD describes a divisor D of F.
If (s = "ideals")
D is given in ideal representation, i.e., divisorD is a list of size 2. divisorD[1] is an ideal of the base ring representing the positive divisor of D and
divisorD[2] is an ideal of the base ring representing the negative divisor. (i.e. D = (I) -(J)).
If (s = "free")
D is given in free representation, i.e., divisorD is a list of places of D. D[i][1] is an prime ideal and D[i][2] an integer, the exponent of the place.

Return: A vector space basis of the Riemann-Roch space of D,
stored in a list RRBasis. The list RRBasis contains a list, say rbasis, and a polynomial, say den. The basis of L(D) consists of all rational functions g/den, where g is an element of rbasis.

Example:

```
LIB "hess.lib";
ring R = 0,(x,y),dp;
poly f  = y^2*(y-1)^3-x^5;
list A1 = list(ideal(x,y-1),1),2;
list A2 = list(ideal(y^2-y+1,x),1),3;
list A3 =  list(ideal(1,y-x),x),-2;
list D = A1,A2;
list E = list(A3);
RiemannRochHess(f,list(D,E),"free");
↦ // ** redefining Infin (    list Infin = maxorderInfinite(f_aff);) hess.l\
   ib::RiemannRochHess:171
↦ [1]:
↦    [1]:
↦       3x5+x3y2-4x2y3-x3y+4x2y2+3y4-6y3+3y2
↦    [2]:
↦       5x3y2-5x2y3-2x3y+5x2y2
↦    [3]:
↦       5x4y-5x3y2+3x3y
↦    [4]:
↦       5x5-5x4y+3x3y
↦    [5]:
↦       x5+2x4y-3xy4+6xy3-3xy2
↦ [2]:
↦    x5
ring S = 0,(x,y,z),dp;
poly f = y^2*(y-1)^3-x^5;
f  = homog(f,z);
ideal P1 = x,y-z;
ideal P2 = y^2-yz+z^2,x;
ideal P3 = x-y,z;
list B1 = P1,2;
list B2 = P2,3;
list B3 = P3,-2;
list Ddivisor = B1,B2,B3;
RiemannRochHess(f,Ddivisor,"free");
↦ [1]:
↦    [1]:
↦       3x5+x3y2-4x2y3-x3y+4x2y2+3y4-6y3+3y2
↦    [2]:
↦       5x3y2-5x2y3-2x3y+5x2y2
↦    [3]:
↦       5x4y-5x3y2+3x3y
↦    [4]:
↦       5x5-5x4y+3x3y
↦    [5]:
↦       x5+2x4y-3xy4+6xy3-3xy2
↦ [2]:
↦    x5
ideal I = intersect(P1^2,P2^3);
ideal J = P3^2;
RiemannRochHess(f,list(I,J),"ideals");
↦ [1]:
↦    [1]:
↦       2x5-x3y2-x2y3+x2y2+2y4-4y3+2y2
↦    [2]:
↦       5x3y2-5x2y3-2x3y+5x2y2
↦    [3]:
↦       5x4y-5x3y2+3x3y
↦    [4]:
↦       5x5-5x4y+3x3y
↦    [5]:
↦       x5+2x4y-3xy4+6xy3-3xy2
↦ [2]:
↦    x5
```

D.5.8 numerAlg_lib

Todos/Issues:

does not follow the naming convention
syntax errors in examples
no test suite

Library: NumerAlg.lib

Purpose: Numerical Algebraic Algorithm

Overview: The library contains procedures to
test the inclusion, the equality of two ideals defined by polynomial systems, compute the degree of a pure i-dimensional component of an algebraic variety defined by a polynomial system,
compute the local dimension of an algebraic variety defined by a polynomial system at a point computed as an approximate value. The use of the library requires to install Bertini (http://www.nd.edu/~sommese/bertini).

Author: Shawki AlRashed, rashed@mathematik.uni-kl.de; sh.shawki@yahoo.de

Procedures:

D.5.8.1 Incl

Procedure from library `numerAlg.lib` (see Section D.5.8 [numerAlg_lib], page 1450).

Usage: Incl(ideal I, ideal J); I, J ideals

Return: t=1 if the algebraic variety defined by I contains the algebraic variety defined by J, otherwise t=0

Example:

```
LIB "numerAlg.lib";
ring r=0,(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal I=f1,f2,f3;
poly g1=(x2+y2+z2-6)*(x-1);
poly g2=(x2+y2+z2-6)*(y-2);
poly g3=(x2+y2+z2-6)*(z-3);
ideal J=g1,g2,g3;
def W=Incl(I,J);
==>
Inclusion:
0
def W=Incl(J,I);
==>
Inclusion:
1
```

D.5.8.2 Equal

Procedure from library `numerAlg.lib` (see Section D.5.8 [numerAlg_lib], page 1450).

Usage: Equal(ideal I, ideal J); I, J ideals

Return: t=1 if the algebraic variety defined by I equals to the algebraic variety defined by J, otherwise t=0

Example:

```
LIB "numerAlg.lib";
ring r=0,(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal I=f1,f2,f3;
poly g1=(x2+y2+z2-6)*(x-1);
poly g2=(x2+y2+z2-6)*(y-2);
poly g3=(x2+y2+z2-6)*(z-3);
ideal J=g1,g2,g3;
def W=Equal(I,J);
==>
Equality:
0
def W=Equal(J,J);
==>
Equality:
1
```

D.5.8.3 DegreePure

Procedure from library `numerAlg.lib` (see Section D.5.8 [numerAlg_lib], page 1450).

Usage: DegreePure(ideal I,int i); I ideal, i positive integer

Return: the degree of the pure i-dimensional component of the algebraic variety defined by I

Example:

```
LIB "numerAlg.lib";
ring r=0,(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal I=f1,f2,f3;
def W=DegreePure(I,1);
==>
The Degree of Component
3
def W=DegreePure(I,2);
==>
The Degree of Component
2
```

D.5.8.4 NumLocalDim

Procedure from library `numerAlg.lib` (see Section D.5.8 [numerAlg_lib], page 1450).

Usage: NumLocalDim(ideal J, list w, int e); J ideal,
w list of an approximate value of a point v in the algebraic variety defined by J, e integer

Return: the local dimension of the algebraic variety defined by J at v

Example:

```
LIB "numerAlg.lib";
int e=14;
ring r=(complex,e,I),(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal J=f1,f2,f3;
list p0=0.99999999999999+I*0.00000000000001,2,3+I*0.00000000000001;
list p2=1,0.99999999999998,2;
list p1=5+I,4.999999999999998+I,5+I;
def D=NumLocalDim(J,p0,e);
==>
The Local Dimension:
0
def D=NumLocalDim(J,p1,e);
==>
The Local Dimension:
1
def D=NumLocalDim(J,p2,e);
==>
The Local Dimension:
2
```

D.5.9 numerDecom_lib

Todos/Issues:

does not follow the naming convention

syntax errors in examples

no test suite

Library: NumDecom.lib

Purpose: Numerical Decomposition of Ideals

Overview: The library contains procedures to compute
numerical irreducible decomposition, and
numerical primary decomposition of an algebraic variety defined by a polynomial system. The use of the library requires to install Bertini (@uref{http://www.nd.edu/~sommese/bertini}).

Author: Shawki AlRashed, rashed@mathematik.uni-kl.de; sh.shawki@yahoo.de

Procedures:

D.5.9.1 re2squ

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: re2squ(ideal I);I ideal

Return: ideal J defined by the polynomial system of the same number of polynomials and unknowns

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
ideal I= x3+y4,z4+yx,xz+3x,x2y+z;
def D=re2squ(I);
setring D;
J;
```

D.5.9.2 UseBertini

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: UseBertini(ideal H,string sv);
H ideal, sv string of the variable of ring

Return: text file called input used in Bertini to compute the solution of the homotopy function H that existed in file input.text

Note: Need to define a start solution of H

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
ideal I= x3+y4,z4+yx,xz+3x,x2y+z;
string sv=varstr(basering);
def A=UseBertini(I,sv);
```

D.5.9.3 Singular2bertini

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: Singular2bertini(list L);L a list

Return: text file called start

Note: adopting the list L to be as a start solution of the homptopy function in Bertini

Example:

```
LIB "numerDecom.lib";
ring r=(complex,16,I),(x,y,z),dp;
list L=list(1,2,3),list(4,5,6+I*2);
def D=Singular2bertini(L);
```

D.5.9.4 bertini2Singular

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: bertini2Singular(string snp, int q);
snp string, q=nvars(basering) integer

Return: re list of the solutions of the homotopy function computed by Bertini

Example:

```
LIB "numerDecom.lib";
ring r = 0,(a,b,c),ds;
int q=nvars(basering);
def T=bertini2Singular("nonsingular_solutions",q);
re;
```

D.5.9.5 ReJunkUseHomo

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: ReJunkUseHomo(ideal I, ideal L, list W, list w);
I ideal, L list of generic linear polynomials {l_1,...,l_i}, W list of a subset of the solution set of the generic slicing V(L) with V(J), w list of a point in V(J)

Return: t=1 if w on an i-dimensional component of V(I), otherwise t=0. Where i=size(L)

Example:

```
LIB "numerDecom.lib";
ring r=(complex,16,I),(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal J=f1,f2,f3;
poly l1=15x+16y+6z+17;
poly l2=2x+14y+4z+18;
ideal L=l1,l2;
list W1=list(0.5372775295412116,-0.7105339291010922,-2.2817700129167831+I*0),list(0.0
list w=list(2,2,-131666666/10000000);
def D=ReJunkUseHomo(J,L,W1,w);
setring D;
t;
```

D.5.9.6 JuReTopDim

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Return: t=1 if w on a d-dimensional component of V(I), otherwise t=0.

Example:

```
LIB "numerDecom.lib";
ring r=(complex,16,I),(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal J=f1,f2,f3;
list w=list(0.5372775295412116,-0.7105339291010922,-2.2817700129167831);
def D=JuReTopDim(J,w,2,2);
setring D;
t;
```

D.5.9.7 JuReZeroDim

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Return: t=1 if w on a positive-dimensional component of V(I), i.e w is not isolated point in V(J)

Example:

```
LIB "numerDecom.lib";
ring r=(complex,16,I),(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal J=f1,f2,f3;
list w1=list(0.5372775295412116,-0.7105339291010922,-2.2817700129167831);
def D1=JuReZeroDim(J,w1,2);
setring D1;
t;
```

D.5.9.8 WitSupSet

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: WitSupSet(ideal I);I ideal

Return: list of Witness point Super Sets W(i) for i=1,...,dim(V(I)), L list of generic linear polynomials and N(0) list of a polynomial system of the same number of polynomials and unknowns. // if W(i) = x, then V(I) has no component of dimension i

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal I=f1,f2,f3;
def W=WitSupSet(I);
setring W;
W(2);
// witness point super set of a pure 2-dimensional component of V(I)
W(1);
// witness point super set of a pure 1-dimensional component of V(I)
W(0);
// witness point super set of a pure 0-dimensional component of V(I)
L;
// list of generic linear polynomials
```

D.5.9.9 WitSet

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: WitSet(ideal I); I ideal

Return: lists W(0..d) of witness point sets of i-dimensional components of V(J) for i=0,...d respectively, where d the dimension of V(J), L list of generic linear polynomials

Note: if W(i)=x, then V(J) has no component of dimension i

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
poly f1=(x3+z)*(x2-y);
poly f2=(x3+y)*(x2-z);
poly f3=(x3+y)*(x3+z)*(z2-y);
ideal I=f1,f2,f3;
def W=WitSet(I);
setring W;
W(1);
// witness point  set of a pure 1-dimensional component of V(I)
W(0);
// witness point  set of a pure 0-dimensional component of V(I)
L;
// list of generic linear polynomials
```

D.5.9.10 NumIrrDecom

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Return: w(1),..., w(t) lists of irreducible witness point sets of irreducible components of V(J)

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
poly f1=(x2+y2+z2-6)*(x-y)*(x-1);
poly f2=(x2+y2+z2-6)*(x-z)*(y-2);
poly f3=(x2+y2+z2-6)*(x-y)*(x-z)*(z-3);
ideal I=f1,f2,f3;
list W=NumIrrDecom(I);
==>
Dimension
0
Number of Components
1
Dimension
1
Number of Components
3
Dimension
2
Number of Components
1
def A(0)=W[1];
// corresponding 0-dimensional components
setring A(0);
w(1);
// corresponding 0-dimensional irreducible component
==> 0-Witness point set (one point)
def A(1)=W[2];
// corresponding 1-dimensional components
setring A(1);
w(1);
// corresponding 1-dimensional irreducible component
==> 1-Witness point set (one point)
w(2);
// corresponding 1-dimensional irreducible component
==> 1-Witness point set (one point)
w(3);
// corresponding 1-dimensional irreducible component
==> 1-Witness point set (one point)
def A(2)=W[3];
// corresponding 2-dimensional components
setring A(2);
w(1);
// corresponding 2-dimensional irreducible component
==> 1-Witness point set (two points)
```

D.5.9.11 defl

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: defl(ideal I, int d); I ideal, int d order of the deflation

Return: deflation ideal DI of I

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y,z),dp;
poly f1=z^2;
poly f2=z*(x^2+y);
ideal I=f1,f2;
def D=defl(I,1);
setring D;
DI;
```

D.5.9.12 NumPrimDecom

Procedure from library `numerDecom.lib` (see Section D.5.9 [numerDecom_lib], page 1453).

Usage: NumPrimDecom(ideal I,int d); I ideal, d order of the deflation

Return: lists of the numerical primary decomposition

Example:

```
LIB "numerDecom.lib";
ring r=0,(x,y),dp;
poly f1=yx;
poly f2=x2;
ideal I=f1,f2;
def W=NumPrimDecom(I,1);
setring W;
w(1);
==> 1-Witness point set (one point)
w(2);
==> 1-Witness point set (one point)
```

D.5.10 orbitparam_lib

Library: orbitparam.lib

Purpose: Parametrizing orbits of unipotent actions

Authors: J. Boehm, boehm at mathematik.uni-kl.de
S. Papadakis, papadak at math.ist.utl.pt

Overview: This library implements the theorem of Chevalley-Rosenlicht as stated in Theorem 3.1.4 of [Corwin, Greenleaf]. Given a set of strictly upper triangular n x n matrices L_1,...,L_c which generate a Lie algebra as a vector space, and a vector v of size n, the function `parametrizeOrbit` constructs a parametrization of the orbit of v under the action of exp(<L_1,...,L_c>).

To compute exp of the Lie algebra elements corresponding to the parameters we require that the characteristic of the base field is zero or larger than n.

By determining the parameters from bottom to top
this allows you to find an element in the orbit with (at least) as many zeros as the dimension of the orbit.

Note: Theorem 3.1.4 of [Corwin, Greenleaf] uses strictly lower triangular matrices.

References:

Laurence Corwin, Frederick P. Greenleaf: Representations of Nilpotent Lie Groups and their Applications: Volume 1, Part 1, Basic Theory and Examples, Cambridge University Press (2004).

Procedures:

D.5.10.1 tangentGens

Procedure from library `orbitparam.lib` (see Section D.5.10 [orbitparam_lib], page 1458).

Usage: tangentGens(L,v); L list, v matrix.

Assume: L is a list of strictly upper triangular n x n matrices of same size. The vector space <L> genererated by the elements of L should be closed under the Lie bracket.

v is matrix of constants of size n x 1.

Return: list, with four entries
- first entry is the dimension of the orbit of under the action of exp(<L>)
- second entry is a list generators of the tangent space of the orbit of v at v under the action of exp(<L>). If the characteristic p of the ground field is positive, then n has to be smaller than p. The generators are elements of <L>.
- third entry is the list of matrices with the coefficient to obtain the generators as a linear combination of the elements of L
- fourth entry is list of integers with entries in v which can be made zero by the action of exp(<L>)

Theory: We apply the theorem of Chevalley-Rosenlicht.

Example:

```
LIB "orbitparam.lib";
ring R = 0,(x),dp;
matrix L1[3][3] = 0,1,0, 0,0,0, 0,0,0;
matrix L2[3][3] = 0,0,1, 0,0,0, 0,0,0;
matrix L3[3][3] = 0,1,1, 0,0,1, 0,0,0;
list L = L1,L2,L3;
matrix v[3][1] = 1,2,3;
tangentGens(L,v);
↦ [1]:
↦    2
↦ [2]:
↦    [1]:
↦       _[1,1]=0
↦       _[1,2]=1/3
↦       _[1,3]=1/3
↦       _[2,1]=0
↦       _[2,2]=0
↦       _[2,3]=1/3
↦       _[3,1]=0
↦       _[3,2]=0
↦       _[3,3]=0
↦    [2]:
↦       _[1,1]=0
↦       _[1,2]=1/2
↦       _[1,3]=0
↦       _[2,1]=0
↦       _[2,2]=0
↦       _[2,3]=0
↦       _[3,1]=0
↦       _[3,2]=0
↦       _[3,3]=0
↦ [3]:
↦    [1]:
↦       _[1,1]=0
↦       _[2,1]=0
↦       _[3,1]=1/3
↦    [2]:
↦       _[1,1]=1/2
↦       _[2,1]=0
↦       _[3,1]=0
↦ [4]:
↦    [1]:
↦       2
↦    [2]:
↦       1
```

D.5.10.2 matrixExp

Procedure from library `orbitparam.lib` (see Section D.5.10 [orbitparam_lib], page 1458).

Usage: matrixExp(A); A matrix.

Assume: A is a nilpotent n x n matrix.
If the characteristic p of the ground field is positive, then n has to be smaller than p.

Return: matrix, exp(A)

Theory: We compute the power series, which terminates since A is nilpotent.

Example:

```
LIB "orbitparam.lib";
ring R = 0,(x),dp;
matrix A[4][4] = 0,0,1,0, 0,0,1,0, 0,0,0,0;
matrixExp(A);
↦ _[1,1]=1
↦ _[1,2]=0
↦ _[1,3]=1
↦ _[1,4]=0
↦ _[2,1]=0
↦ _[2,2]=1
↦ _[2,3]=1
↦ _[2,4]=0
↦ _[3,1]=0
↦ _[3,2]=0
↦ _[3,3]=1
↦ _[3,4]=0
↦ _[4,1]=0
↦ _[4,2]=0
↦ _[4,3]=0
↦ _[4,4]=1
```

D.5.10.3 matrixLog

Procedure from library `orbitparam.lib` (see Section D.5.10 [orbitparam_lib], page 1458).

Usage: matrixLog(A); A matrix.

Assume: A-E is a nilpotent n x n matrix.
If the characteristic p of the ground field is positive, then n has to be smaller than p.

Return: matrix, log(A)

Theory: We compute the power series, which terminates since A-E is nilpotent.

Example:

```
LIB "orbitparam.lib";
ring R = 0,(s,t),dp;
matrix A[3][3] = 1,s,st/2, 0,1,t, 0,0,1;
matrixLog(A);
↦ _[1,1]=0
↦ _[1,2]=s
↦ _[1,3]=0
↦ _[2,1]=0
↦ _[2,2]=0
↦ _[2,3]=t
↦ _[3,1]=0
↦ _[3,2]=0
↦ _[3,3]=0
```

D.5.10.4 parametrizeOrbit

Procedure from library `orbitparam.lib` (see Section D.5.10 [orbitparam_lib], page 1458).

Usage: parametrizeOrbit(L,v); L list, v matrix.

Assume: L is a list of strictly upper triangular n x n matrices of same size. The vector space <L> genererated by the elements of L should be closed under the Lie bracket.
v is matrix of constants of size n x 1.
The basering has at least size(L) variables. However we will only use tangentGens(L,v)[1] many of them.

Return: list, with four entries
- int, dimension of the orbit
- matrix A over the basering giving a parametrization of the orbit of v under the action of exp(<L>).
- list of integers, with the (row)-indices of entries which can be deleted by the action
- the variables of the parametrization to solve for

Theory: We apply the theorem of Chevalley-Rosenlicht. First we determine tangent space generators, then apply `matrixExp` to the generators, and finally take the product to obtain the parametrization.

Example:

```
LIB "orbitparam.lib";
ring R = 0,(t(1..3)),dp;
matrix L1[3][3] = 0,1,0, 0,0,0, 0,0,0;
matrix L2[3][3] = 0,0,1, 0,0,0, 0,0,0;
matrix L3[3][3] = 0,1,1, 0,0,1, 0,0,0;
list L = L1,L2,L3;
matrix v[3][1] = 1,2,3;
parametrizeOrbit(L,v);
↦ [1]:
↦    2
↦ [2]:
↦    _[1,1]=1/6*t(1)^2+5/3*t(1)+t(2)+1
↦    _[2,1]=t(1)+2
↦    _[3,1]=3
↦ [3]:
↦    [1]:
↦       2
↦    [2]:
↦       1
↦ [4]:
↦    [1]:
↦       t(1)
↦    [2]:
↦       t(2)
ring R1 = 0,(t(1..2)),dp;
matrix L1[4][4] = 0,1,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,0;
matrix L2[4][4] = 0,0,1,0, 0,0,0,1, 0,0,0,0, 0,0,0,0;
list L = L1,L2;
matrix v[4][1] = 1,2,3,4;
parametrizeOrbit(L,v);
↦ [1]:
↦    2
↦ [2]:
↦    _[1,1]=1/4*t(1)*t(2)+1/2*t(1)+3/4*t(2)+1
↦    _[2,1]=t(2)+2
↦    _[3,1]=t(1)+3
↦    _[4,1]=4
↦ [3]:
↦    [1]:
↦       3
↦    [2]:
↦       2
↦ [4]:
↦    [1]:
↦       t(1)
↦    [2]:
↦       t(2)
```

D.5.10.5 maxZeros

Procedure from library `orbitparam.lib` (see Section D.5.10 [orbitparam_lib], page 1458).

Usage: maxZeros(L,v); L list, v matrix.

Assume: L is a list of strictly upper triangular n x n matrices of same size. The vector space <L> genererated by the elements of L should be closed under the Lie bracket.

v is matrix of constants of size n x 1.

The basering has at least size(L) variables. However we will only use tangentGens(L,v)[1] many of them.

Return: matrix of constants over the basering giving an element in the orbit of v under the action of exp(<L>) with (at least) as many zeros as the dimension of the orbit.

Theory: We apply `parametrizeOrbit` to obtain a parametrization of the orbit according to the theorem of Chevalley-Rosenlicht. By determining the parameters from bottom to top we find an element in the orbit with (at least) as many zeros as the dimension of the orbit.

Example:

```
LIB "orbitparam.lib";
ring R = 0,(x),dp;
matrix L1[3][3] = 0,1,0, 0,0,0, 0,0,0;
matrix L2[3][3] = 0,0,1, 0,0,0, 0,0,0;
matrix L3[3][3] = 0,1,1, 0,0,1, 0,0,0;
list L = L1,L2,L3;
matrix v[3][1] = 1,2,3;
maxZeros(L,v);
↦ _[1,1]=0
↦ _[2,1]=0
↦ _[3,1]=3
ring R1 = 0,(x),dp;
matrix L1[4][4] = 0,1,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,0;
matrix L2[4][4] = 0,0,1,0, 0,0,0,1, 0,0,0,0, 0,0,0,0;
list L = L1,L2;
matrix v[4][1] = 1,2,3,4;
maxZeros(L,v);
↦ _[1,1]=-1/2
↦ _[2,1]=0
↦ _[3,1]=0
↦ _[4,1]=4
```

D.5.11 paraplanecurves_lib

Library: paraplanecurves.lib

Purpose: Rational parametrization of rational plane curves

Authors: J. Boehm, boehm at mathematik.uni-kl.de
W. Decker, decker at mathematik.uni-kl.de
S. Laplagne, slaplagn at dm.uba.ar
F. Seelisch, seelisch at mathematik.uni-kl.de

Overview: Suppose C = {f(x,y,z)=0} is a rational plane curve, where f is homogeneous of degree n with coefficients in Q and absolutely irreducible (these conditions are checked automatically.)
After a first step, realized by a projective automorphism in the procedure adjointIdeal, C satisfies:
- C does not have singularities at infinity z=0.

- C does not contain the point (0:1:0) (that is, the dehomogenization of f with respect to z is monic as a polynomial in y).
Considering C in the chart z<>0, the algorithm regards x as transcendental and y as algebraic and computes an integral basis in C(x)[y] of the integral closure of C[x] in C(x,y) using the normalization algorithm from Section D.4.23 [normal_lib], page 1183: see Section D.4.12 [integralbasis_lib], page 1114. In a future edition of the library, also van Hoeij's algorithm for computing the integral basis will be available.
From the integral basis, the adjoint ideal is obtained by linear algebra. Alternatively, the algorithm starts with a local analysis of the singular locus of C. Then, for each primary component of the singular locus which does not correspond to ordinary multiple points or cusps, the integral basis algorithm is applied separately. The ordinary multiple points and cusps, in turn, are addressed by a straightforward direct algorithm. The adjoint ideal is obtained by intersecting all ideals obtained locally. The local variant of the algorithm is used by default.
The linear system corresponding to the adjoint ideal maps the curve birationally to a rational normal curve in P^(n-2).
Iterating the anticanonical map, the algorithm projects the rational normal curve to PP1 for n odd resp. to a conic C2 in PP2 for n even.
In case n is even, the algorithm tests whether there is a rational point on C2 and if so gives a parametrization of C2 which is defined over Q. Otherwise, the parametrization given is defined over a quadratic field extension of Q.
By inverting the birational map of C to PP1 resp. to C2, a parametrization of C is obtained (defined over Q or the quadratic field extension).

References:

Janko Boehm: Parametrisierung rationaler Kurven, Diploma Thesis, http://www.math.uni-sb.de/ag/schreyer/jb/diplom%20janko%20boehm.pdf

Janko Boehm, Wolfram Decker, Santiago Laplagne, Gerhard Pfister: Local to global algorithms for the Gorenstein adjoint ideal of a curve, Algorithmic and Experimental Methods in Algebra, Geometry, and Number Theory, Springer 2018

Theo de Jong: An algorithm for computing the integral closure, Journal of Symbolic Computation 26 (3) (1998), p. 273-277

Gert-Martin Greuel, Santiago Laplagne, Frank Seelisch: Normalization of Rings, Journal of Symbolic Computation 9 (2010), p. 887-901

Mark van Hoeij: An Algorithm for Computing an Integral Basis in an Algebraic Function Field, Journal of Symbolic Computation 18 (1994), p. 353-363, http://www.math.fsu.edu/~hoeij/papers/comments/jsc1994.html

Procedures:

D.5.11.1 adjointIdeal

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: adjointIdeal(f [, choices]); f polynomial in three variables, choices optional list consisting of one integer or of one string or of one integer followed by one string.
Optional integer can be:
1: compute integral basis via normalization.
2: make local analysis of singularities first and apply normalization separately.
3: normalization via ideal quotient.

4: normalization via local ideal quotient.
The default is 2.
Optional string may contain substrings:
- rattestyes -> causes error message if curve is not rational.
- firstchecksdone -> prevents that check of assumptions will be done more than once.

Assume: The basering must be a polynomial ring in three variables, say x,y,z, with coefficients in Q.
The polynomial f must be homogeneous and absolutely irreducible.
All these conditions will be checked automatically.

Return: ideal, the adjoint ideal of the curve defined by f.

Theory: Considering C in the chart z<>0, the algorithm regards x as transcendental and y as algebraic and computes an integral basis in C(x)[y] of the integral closure of C[x] in C(x,y) using the normalization algorithm from Section D.4.23 [normal_lib], page 1183: see Section D.4.12 [integralbasis_lib], page 1114. In a future edition of the library, also van Hoeij's algorithm for computing the integral basis will be available.
From the integral basis, the adjoint ideal is obtained by linear algebra. Alternatively, the algorithm starts with a local analysis of the singular locus of C. Then, for each primary component of the singular locus which does not correspond to ordinary multiple points or cusps, the integral basis algorithm is applied separately. The ordinary multiple points and cusps, in turn, are addressed by a straightforward direct algorithm. The adjoint ideal is obtained by intersecting all ideals obtained locally. The local variant of the algorithm is used by default.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
adjointIdeal(f);
↦ _[1]=y6
↦ _[2]=xy5+y5z
↦ _[3]=x2y4+xy4z
↦ _[4]=x3y3+2x2y3z+xy3z2
↦ _[5]=x4y2+3x3y2z+3x2y2z2+xy2z3
↦ _[6]=x5y+3x4yz+3x3yz2+x2yz3
↦ _[7]=x6+4x5z+6x4z2+4x3z3+x2z4
```

D.5.11.2 invertBirMap

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: invertBirMap(phi, I); phi ideal, I ideal

Assume: The ideal phi in the basering R represents a birational map of the variety given by the ideal I in R to its image in projective space P = PP^(size(phi)-1).

Note: The procedure might fail or give a wrong output if phi does not define a birational map.

Return: ring, the coordinate ring of P, with an ideal named J and an ideal named psi.
The ideal J defines the image of phi.

The ideal psi gives the inverse of phi.
Note that the entries of psi should be considered as representatives of classes in the quotient ring R/J.

Theory: We compute the ideal I(G) in R**S of the graph G of phi.
The ideal J is given by the intersection of I(G) with S.
The map psi is given by a relation mod J of those relations in I(G) which are linear in the variables of R.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
ideal adj = adjointIdeal(f);
def Rn = invertBirMap(adj,ideal(f));
↦ // 'invertBirMap' created a ring together with two ideals J and psi.
↦ // Supposing you typed, say,  def RPn = invertBirMap(phi,I);
↦ // you may access the ideals by typing
↦ //      setring RPn; J; psi;
setring(Rn);
J;
↦ J[1]=y(5)*y(6)-y(4)*y(7)
↦ J[2]=y(4)*y(6)-y(3)*y(7)
↦ J[3]=y(2)*y(6)-y(1)*y(7)
↦ J[4]=y(4)*y(5)-y(2)*y(7)
↦ J[5]=y(3)*y(5)-y(1)*y(7)
↦ J[6]=y(1)*y(5)-y(7)^2
↦ J[7]=y(4)^2-y(1)*y(7)
↦ J[8]=y(3)*y(4)-y(1)*y(6)
↦ J[9]=y(2)*y(4)-y(7)^2
↦ J[10]=y(1)*y(4)-y(6)*y(7)
↦ J[11]=y(2)*y(3)-y(6)*y(7)
↦ J[12]=y(1)*y(3)-y(6)^2
↦ J[13]=y(2)^2-y(5)*y(7)
↦ J[14]=y(1)*y(2)-y(4)*y(7)
↦ J[15]=y(1)^2-y(3)*y(7)
↦ J[16]=y(1)*y(6)^2-y(3)^2*y(7)
↦ J[17]=y(6)^4-y(3)^3*y(7)
psi;
↦ psi[1]=-y(6)^2
↦ psi[2]=-y(4)*y(7)
↦ psi[3]=y(6)^2-y(5)*y(7)
```

D.5.11.3 paraPlaneCurve

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: paraPlaneCurve(f [, s]); f poly , s optional string
optional string s can be:
'normal': compute integral basis via normalization.
'local': make local analysis of singularities first and apply normalization separately.

The default is 2.

Assume: The basering must be a polynomial ring in three variables, say x,y,z, with coefficients in Q.
The polynomial f must be homogeneous and absolutely irreducible.
The curve C = {f = 0} must be rational, i.e., have geometric genus 0 (see Section D.4.23.7 [genus], page 1200).
These conditions will be checked automatically.

Return: ring with an ideal PARA which contains a rational parametrization of the rational plane curve given by f; the ground field of the returned polynomial ring is either Q or some algebraic extension Q(a); PARA consists of three generators that parametrize the three coordinates of the rational curve

Theory: After a first step, realized by a projective automorphism in the procedure adjointIdeal, C satisfies:
- C does not have singularities at infinity z=0.
- C does not contain the point (0:1:0) (that is, the dehomogenization of f with respect to z is monic as a polynomial in y).
Considering C in the chart z<>0, the algorithm regards x as transcendental and y as algebraic and computes an integral basis in C(x)[y] of the integral closure of C[x] in C(x,y) using the normalization algorithm from Section D.4.23 [normal_lib], page 1183: see Section D.4.12 [integralbasis_lib], page 1114. In a future edition of the library, also van Hoeij's algorithm for computing the integral basis will be available.
From the integral basis, the adjoint ideal is obtained by linear algebra. Alternatively, the algorithm starts with a local analysis of the singular locus of C. Then, for each primary component of the singular locus which does not correspond to ordinary multiple points or cusps, the integral basis algorithm is applied separately. The ordinary multiple points and cusps, in turn, are addressed by a straightforward direct algorithm. The adjoint ideal is obtained by intersecting all ideals obtained locally. The local variant of the algorithm is used by default.
The linear system corresponding to the adjoint ideal maps the curve birationally to a rational normal curve in P^(n-2).
Iterating the anticanonical map, the algorithm projects the rational normal curve to PP1 for n odd resp. to a conic C2 in PP2 for n even.
In case n is even, the algorithm tests whether there is a rational point on C2 and if so gives a parametrization of C2 which is defined over Q. Otherwise the parametrization is defined over a quadratic field extension of Q.
By inverting the birational map of C to PP1 resp. to C2, a parametrization of C is obtained (defined over Q or the quadratic field extension).

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f1 = 1/2*x^5+x^2*y*z^2+x^3*y*z+1/2*x*y^2*z^2-2*x*y^3*z+y^5;
def Rp1 = paraPlaneCurve(f1);
↦ // 'paraPlaneCurve' created a ring together with an ideal PARA.
↦ // Supposing you typed, say,  def RP1 = paraPlaneCurve(f);
↦ // you may access the ideal by typing
↦ //      setring RP1; PARA;
setring Rp1;
PARA;
```

```
↦ PARA[1]=-4s4t+2s3t2
↦ PARA[2]=2s2t3-st4
↦ PARA[3]=4s5-t5
setring R;
poly f2 = x6+3x4y2+3x2y4+y6-4x4z2-34x3yz2-7x2y2z2+12xy3z2+6y4z2;
f2 = f2+36x2z4+36xyz4+9y2z4;
def Rp2 = paraPlaneCurve(f2);
↦ // 'paraPlaneCurve' created a ring together with an ideal PARA.
↦ // Supposing you typed, say,  def RP1 = paraPlaneCurve(f);
↦ // you may access the ideal by typing
↦ //      setring RP1; PARA;
setring Rp2;
PARA;
↦ PARA[1]=(-a)*s6+(-3821910a+568836)*s5t+(-17261814423635a-17036342693900)*\
   s4t2+(-40791433831085325700a-41443502254869224120)*s3t3+(7095798487219992\
   1706914985a-63670892810117870548425400)*s2t4+(1193953013667763912950254333\
   032250a-70343024051936434031711433716044)*st5+(14536738745802049687386582\
   3959936915531a+80558165687949544164912330361180107460)*t6
↦ PARA[2]=s6+(568836a+3821910)*s5t+(20661581653300a+24799008638835)*s4t2+(5\
   74668320454005486 80a+45699794797119440900)*s3t3+(-821403224969470541096766\
   600a+33367675319234688316737815)*s2t4+(-15376445724248718869904934835284 4\
   a+37012366361272671947391736404550)*st5+(-15895168204498579954177123982 45\
   02169340a+12526468076865336379874666653301093 1669)*t6
↦ PARA[3]=(90983469924655936000a+37676837794369664000)*s3t3+(18458128762166\
   9915274432000a+46121405018308889263872000)*s2t4+(121948097419855203536634\
   716352000a+11879229621327449840239705728000)*st5+(26268443692822012344779\
   010112242496000a-13315067478818422633363673217725 44000)*t6
```

D.5.11.4 rncAntiCanonicalMap

Procedure from library paraplanecurves.lib (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: rncAntiCanonicalMap(I); I ideal

Assume: I is a homogeneous ideal in the basering
defining a rational normal curve C in PP^n.

Note: The procedure will fail or give a wrong output if I is not the ideal of a rational normal curve.

Return: ideal defining the anticanonical map C –> PP^(n-2).
Note that the entries of the ideal should be considered as representatives of elements in R/I, where R is the basering.

Theory: The anti-canonical map of a rational normal curve
maps C isomorpically to a rational normal curve in PP^(n-2).

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
ideal adj = adjointIdeal(f);
def Rn = mapToRatNormCurve(f,adj);
↦ //'mapToRatNorm' created a ring together with an ideal RNC.
↦ // Supposing you typed, say,  def RPn = mapToRatNorm(f,AI);
↦ // you may access the ideal by typing
↦ //      setring RPn; RNC;
setring(Rn);
RNC;
↦ RNC[1]=y(5)*y(6)-y(4)*y(7)
↦ RNC[2]=y(4)*y(6)-y(3)*y(7)
↦ RNC[3]=y(2)*y(6)-y(1)*y(7)
↦ RNC[4]=y(4)*y(5)-y(2)*y(7)
↦ RNC[5]=y(3)*y(5)-y(1)*y(7)
↦ RNC[6]=y(1)*y(5)-y(7)^2
↦ RNC[7]=y(4)^2-y(1)*y(7)
↦ RNC[8]=y(3)*y(4)-y(1)*y(6)
↦ RNC[9]=y(2)*y(4)-y(1)*y(5)
↦ RNC[10]=y(1)*y(4)-y(6)*y(7)
↦ RNC[11]=y(2)*y(3)-y(6)*y(7)
↦ RNC[12]=y(1)*y(3)-y(6)^2
↦ RNC[13]=y(2)^2-y(5)*y(7)
↦ RNC[14]=y(1)*y(2)-y(4)*y(7)
↦ RNC[15]=y(1)^2-y(3)*y(7)
↦ RNC[16]=y(1)*y(6)^2-y(3)^2*y(7)
↦ RNC[17]=y(6)^4-y(3)^3*y(7)
rncAntiCanonicalMap(RNC);
↦ _[1]=y(1)
↦ _[2]=-y(2)
↦ _[3]=-y(5)
↦ _[4]=-y(4)
↦ _[5]=-y(7)
```

D.5.11.5 rationalPointConic

Procedure from library paraplanecurves.lib (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: rationalPointConic(p); p poly

Assume: assumes that p is an irreducible quadratic polynomial in the first three ring variables; ground field is expected to be Q.

Return: The method finds a point on the given conic. There are two possibilities:
1) There is a rational point on the curve.
2) There is no rational point on the curve.
In the second case, the method creates a modification of the current basering which is a polynomial ring over some quadratic field extension Q(a) of Q. Apart from the replacement of Q by Q(a), the new polynomial ring, R say, is the same as the original basering. (In the first case, R is identical with the basering.) In both cases, the method will then define a (1x3) matrix named 'point' which lives in R and which contains the coordinates of the desired point on q.
Finally, the method returns the ring R (which will in the 1st case be the original base ring).

Example:

```
LIB "paraplanecurves.lib";
ring R = 0, (x,y,z), dp;
```

```
system("random", 4711);
poly p = x^2 + 2*y^2 + 5*z^2 - 4*x*y + 3*x*z + 17*y*z;
def S = rationalPointConic(p); // quadratic field extension,
// minpoly = a^2 - 2
testPointConic(p, S);
↦ conic: x2-4xy+2y2+3xz+17yz+5z2
↦ point: (-1/4a), (-1/4a+1/4), 0
↦ minpoly: (a2-2)
↦ 1
setring R;
p = x^2 - 1857669520 * y^2 + 86709575222179747132487270400 * z^2;
S = rationalPointConic(p); // same as current basering,
// no extension needed
testPointConic(p, S);
↦ conic: x2-1857669520y2+86709575222179747132487270400z2
↦ point: 819398304609269135405871919690911280, 4998995645707055256041604614\
   16220, 73117135886813712113057
↦ 1
```

D.5.11.6 mapToRatNormCurve

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: mapToRatNormCurve(f, AI); f polynomial, AI ideal

Assume: The polynomial f is homogeneous in three variables and absolutely irreducible.
The plane curve C defined by f is rational.
The ideal AI is the adjoint ideal of C.

Return: ring with an ideal RNC.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
ideal adj = adjointIdeal(f);
def Rn = mapToRatNormCurve(f,adj);
↦ //'mapToRatNorm' created a ring together with an ideal RNC.
↦ // Supposing you typed, say,  def RPn = mapToRatNorm(f,AI);
↦ // you may access the ideal by typing
↦ //      setring RPn; RNC;
setring(Rn);
RNC;
↦ RNC[1]=y(5)*y(6)-y(4)*y(7)
↦ RNC[2]=y(4)*y(6)-y(3)*y(7)
↦ RNC[3]=y(2)*y(6)-y(1)*y(7)
↦ RNC[4]=y(4)*y(5)-y(2)*y(7)
↦ RNC[5]=y(3)*y(5)-y(1)*y(7)
↦ RNC[6]=y(1)*y(5)-y(7)^2
↦ RNC[7]=y(4)^2-y(1)*y(7)
↦ RNC[8]=y(3)*y(4)-y(1)*y(6)
↦ RNC[9]=y(2)*y(4)-y(1)*y(5)
↦ RNC[10]=y(1)*y(4)-y(6)*y(7)
↦ RNC[11]=y(2)*y(3)-y(6)*y(7)
↦ RNC[12]=y(1)*y(3)-y(6)^2
↦ RNC[13]=y(2)^2-y(5)*y(7)
↦ RNC[14]=y(1)*y(2)-y(4)*y(7)
↦ RNC[15]=y(1)^2-y(3)*y(7)
↦ RNC[16]=y(1)*y(6)^2-y(3)^2*y(7)
↦ RNC[17]=y(6)^4-y(3)^3*y(7)
```

D.5.11.7 rncItProjOdd

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: rncItProjOdd(I); I ideal

Assume: I is a homogeneous ideal in the basering with n+1 variables defining a rational normal curve C in PP^n with n odd.

Note: The procedure will fail or give a wrong output if I is not the ideal of a rational normal curve. It will test whether n is odd.

Return: ideal PHI defining an isomorphic projection of C to PP^1.
Note that the entries of PHI should be considered as
representatives of elements in R/I, where R is the basering.

Theory: We iterate the procedure Section D.5.11.4 [rncAntiCanonicalMap], page 1468 to obtain PHI.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = -x7-10x5y2-10x4y3-3x3y4+8x2y5+7xy6+11y7+3x6+10x5y +30x4y2
+26x3y3-13x2y4-29xy5-33y6-3x5-20x4y-33x3y2-8x2y3+37xy4+33y5
+x4+10x3y+13x2y2-15xy3-11y4;
f = homog(f,z);
ideal adj = adjointIdeal(f);
def Rn = mapToRatNormCurve(f,adj);
↦ //'mapToRatNorm' created a ring together with an ideal RNC.
↦ // Supposing you typed, say,  def RPn = mapToRatNorm(f,AI);
↦ // you may access the ideal by typing
↦ //      setring RPn; RNC;
setring(Rn);
RNC;
↦ RNC[1]=y(4)*y(5)-y(3)*y(6)
↦ RNC[2]=y(2)*y(5)-y(1)*y(6)
↦ RNC[3]=y(4)^2-y(2)*y(6)
↦ RNC[4]=y(3)*y(4)-y(1)*y(6)
↦ RNC[5]=11*y(1)*y(4)+7*y(2)*y(4)+8*y(1)*y(6)-3*y(2)*y(6)-10*y(3)*y(6)-10*y\
   (4)*y(6)-y(6)^2
↦ RNC[6]=y(3)^2-y(1)*y(5)
↦ RNC[7]=11*y(2)*y(3)+7*y(2)*y(4)+8*y(1)*y(6)-3*y(2)*y(6)-10*y(3)*y(6)-10*y\
   (4)*y(6)-y(6)^2
↦ RNC[8]=11*y(1)*y(3)+7*y(1)*y(4)+8*y(1)*y(5)-10*y(3)*y(5)-3*y(1)*y(6)-10*y\
   (3)*y(6)-y(5)*y(6)
↦ RNC[9]=121*y(1)*y(2)+77*y(2)^2-89*y(2)*y(4)-174*y(1)*y(6)-86*y(2)*y(6)+80\
   *y(3)*y(6)+69*y(4)*y(6)+8*y(6)^2
```

```
↦ RNC[10]=1331*y(1)^2-539*y(2)^2+1246*y(2)*y(4)-1914*y(1)*y(5)+880*y(3)*y(5\
   )+984*y(1)*y(6)+335*y(2)*y(6)-691*y(3)*y(6)-1373*y(4)*y(6)+88*y(5)*y(6)-1\
   45*y(6)^2
rncItProjOdd(RNC);
↦ _[1]=121*y(3)+77*y(4)
↦ _[2]=-11*y(5)-7*y(6)
```

See also: Section D.5.11.8 [rncItProjEven], page 1472.

D.5.11.8 rncItProjEven

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: rncItProjEven(I); I ideal

Assume: I is a homogeneous ideal in the basering with n+1 variables defining a rational normal curve C in PP^n with n even.

Note: The procedure will fail or give a wrong output if I is not the ideal of a rational normal curve. It will test whether n is odd.

Return: ring with an ideal CONIC defining a conic C2 in PP^2.
In addition, an ideal PHI in the basering defining an isomorphic projection of C to C2 will be exported.
Note that the entries of PHI should be considered as
representatives of elements in R/I, where R is the basering.

Theory: We iterate the procedure Section D.5.11.4 [rncAntiCanonicalMap], page 1468 to obtain PHI.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
ideal adj = adjointIdeal(f);
def Rn = mapToRatNormCurve(f,adj);
↦ //'mapToRatNorm' created a ring together with an ideal RNC.
↦ // Supposing you typed, say,  def RPn = mapToRatNorm(f,AI);
↦ // you may access the ideal by typing
↦ //      setring RPn; RNC;
setring(Rn);
RNC;
↦ RNC[1]=y(5)*y(6)-y(4)*y(7)
↦ RNC[2]=y(4)*y(6)-y(3)*y(7)
↦ RNC[3]=y(2)*y(6)-y(1)*y(7)
↦ RNC[4]=y(4)*y(5)-y(2)*y(7)
↦ RNC[5]=y(3)*y(5)-y(1)*y(7)
↦ RNC[6]=y(1)*y(5)-y(7)^2
↦ RNC[7]=y(4)^2-y(1)*y(7)
↦ RNC[8]=y(3)*y(4)-y(1)*y(6)
↦ RNC[9]=y(2)*y(4)-y(1)*y(5)
↦ RNC[10]=y(1)*y(4)-y(6)*y(7)
↦ RNC[11]=y(2)*y(3)-y(6)*y(7)
↦ RNC[12]=y(1)*y(3)-y(6)^2
↦ RNC[13]=y(2)^2-y(5)*y(7)
↦ RNC[14]=y(1)*y(2)-y(4)*y(7)
↦ RNC[15]=y(1)^2-y(3)*y(7)
↦ RNC[16]=y(1)*y(6)^2-y(3)^2*y(7)
↦ RNC[17]=y(6)^4-y(3)^3*y(7)
def Rc = rncItProjEven(RNC);
PHI;
↦ PHI[1]=-y(7)
↦ PHI[2]=-y(2)
↦ PHI[3]=-y(5)
setring Rc;
CONIC;
↦ y(2)^2-y(1)*y(3)
```

See also: Section D.5.11.7 [rncItProjOdd], page 1471.

D.5.11.9 paraConic

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: paraConic(q); q poly

Assume: The basering must be a polynomial ring in three variables with coefficients in Q. The polynomial q must be homogeneous of degree 2 and absolutely irreducible.

Note: The procedure might fail or give a wrong output if the assumptions do not hold.

Return: ring with an ideal PARACONIC. The ring should be considered as the homogeneous coordinate ring of PP^1, the ideal defines a rational parametrization PP^1 –> C2 = {q=0}.

Theory: We compute a point on C2 via Section D.5.11.5 [rationalPointConic], page 1469. The pencil of lines through this point projects C2 birationally to PP^1. Inverting the projection gives the result.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
ideal adj = adjointIdeal(f);
def Rn = invertBirMap(adj,ideal(f));
↦ // 'invertBirMap' created a ring together with two ideals J and psi.
↦ // Supposing you typed, say,  def RPn = invertBirMap(phi,I);
↦ // you may access the ideals by typing
↦ //      setring RPn; J; psi;
setring(Rn);
J;
↦ J[1]=y(5)*y(6)-y(4)*y(7)
↦ J[2]=y(4)*y(6)-y(3)*y(7)
↦ J[3]=y(2)*y(6)-y(1)*y(7)
↦ J[4]=y(4)*y(5)-y(2)*y(7)
↦ J[5]=y(3)*y(5)-y(1)*y(7)
↦ J[6]=y(1)*y(5)-y(7)^2
↦ J[7]=y(4)^2-y(1)*y(7)
↦ J[8]=y(3)*y(4)-y(1)*y(6)
↦ J[9]=y(2)*y(4)-y(7)^2
↦ J[10]=y(1)*y(4)-y(6)*y(7)
↦ J[11]=y(2)*y(3)-y(6)*y(7)
↦ J[12]=y(1)*y(3)-y(6)^2
↦ J[13]=y(2)^2-y(5)*y(7)
↦ J[14]=y(1)*y(2)-y(4)*y(7)
↦ J[15]=y(1)^2-y(3)*y(7)
↦ J[16]=y(1)*y(6)^2-y(3)^2*y(7)
↦ J[17]=y(6)^4-y(3)^3*y(7)
def Rc = rncItProjEven(J);
PHI;
↦ PHI[1]=-y(7)
↦ PHI[2]=-y(2)
↦ PHI[3]=-y(5)
setring Rc;
CONIC;
↦ y(2)^2-y(1)*y(3)
def RPc = paraConic(CONIC);
↦ // 'paraConic' created a ring together with an ideal RNC.
↦ // Supposing you typed, say,  def RP1 = paraConic(q);
↦ // you may access the ideal by typing
↦ //      setring RP1; PARACONIC;
setring RPc;
PARACONIC;
↦ PARACONIC[1]=s2
↦ PARACONIC[2]=st
↦ PARACONIC[3]=t2
```

See also: Section D.5.11.5 [rationalPointConic], page 1469.

D.5.11.10 testParametrization

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: testParametrization(f, rTT); f poly, rTT ring

Assume: The assumptions on the basering and the polynomial f are as required by Section D.5.11.3 [paraPlaneCurve], page 1466. The ring rTT has two variables and contains an ideal PARA (such as the ring obtained by applying Section D.5.11.3 [paraPlaneCurve], page 1466 to f).

Return: int which is 1 if PARA defines a parametrization of the curve {f=0} and 0, otherwise.

Theory: We compute the polynomial defining the image of PARA
and compare it with f.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0,(x,y,z),dp;
poly f = y^8-x^3*(z+x)^5;
def RP1 = paraPlaneCurve(f);
↦ // 'paraPlaneCurve' created a ring together with an ideal PARA.
↦ // Supposing you typed, say,  def RP1 = paraPlaneCurve(f);
↦ // you may access the ideal by typing
↦ //      setring RP1; PARA;
```

```
testParametrization(f, RP1);
↦ 1
```

D.5.11.11 testPointConic

Procedure from library `paraplanecurves.lib` (see Section D.5.11 [paraplanecurves_lib], page 1463).

Usage: testPointConic(p, r); p poly, r ring

Assume: assumes that p is a homogeneous quadratic polynomial in the first three ring variables of the current basering;
Assumes that there is a (1x3) matrix named 'point' in r with entries from the ground field of r.

Return: returns 1 iff the point named 'point', residing in r, lies on the conic given by p; 0 otherwise

Note: This method temporarily changes the basering to r. Afterwards, the basering will be the same as before.

Example:

```
LIB "paraplanecurves.lib";
ring R = 0, (x,y,z), dp;
system("random", 4711);
poly p = x^2 + 2*y^2 + 5*z^2 - 4*x*y + 3*x*z + 17*y*z;
def S = rationalPointConic(p);
if (testPointConic(p, S) == 1)
↦ conic: x2-4xy+2y2+3xz+17yz+5z2
↦ point: (-1/4a), (-1/4a+1/4), 0
↦ minpoly: (a2-2)
{ "point lies on conic"; }
↦ point lies on conic
else
{ "point does not lie on conic"; }
```

D.5.12 resbinomial_lib

Todos/Issues:

formatting is inappropriate

avoid export

bad names(or should be static): identifyvars, elimrep, convertdata, lcmofall, genoutput, salida, iniD, reslist, sumlist, dividelist, createlist

Library: resbinomial.lib

Purpose: Combinatorial algorithm of resolution of singularities of binomial ideals in arbitrary characteristic. Binomial resolution algorithm of Blanco

Authors: R. Blanco, mariarocio.blanco@uclm.es,
G. Pfister, pfister@mathematik.uni-kl.de

Procedures:

D.5.12.1 BINresol

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: BINresol(J); J ideal

Return: E-resolution of singularities of a binomial ideal J in terms of the affine charts, see example

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^2-x(2)^3;
list B=BINresol(J);
B[1]; // list of final charts
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [2]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [3]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2)
↦ //        block   2 : ordering C
↦ [4]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2)
↦ //        block   2 : ordering C
B[2]; // list of all charts
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [2]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [3]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [4]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [5]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [6]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2)
↦ //        block   2 : ordering C
↦ [7]:
↦    // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2)
↦ //        block   2 : ordering C
ring r = 2,(x(1..3)),dp;
↦ // ** redefining r (ring r = 2,(x(1..3)),dp;) ./examples/BINresol.sing:7
ideal J=x(1)^2-x(2)^2*x(3)^2;
list B=BINresol(J);
↦ // ** redefining B (list B=BINresol(J);) ./examples/BINresol.sing:9
B[2]; // list of all charts
↦ [1]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [2]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [3]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [4]:
```

```
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [5]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [6]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [7]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [8]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [9]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) x(3)
↦ //        block   2 : ordering C
↦ [10]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [11]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) y(3)
↦ //        block   2 : ordering C
↦ [12]:
↦    // coefficients: ZZ/2
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2) x(3)
↦ //        block   2 : ordering C
```

D.5.12.2 Eresol

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Eresol(J); J ideal

Return: The E-resolution of singularities of J in terms of the affine charts, see example

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^2-x(2)^3;
list L=Eresol(J);
"Please press return after each break point to see the next element of the output lis
↦ Please press return after each break point to see the next element of the\
    output list
L[1][1]; // information of the first chart, L[1] list of charts
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    0
↦ [4]:
↦    [1]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             3
↦       [2]:
↦          [1]:
↦             2
↦          [2]:
↦             0
↦ [5]:
↦    [1]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦ [6]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦ [7]:
↦    0
↦ [8]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦ [9]:
↦    _[1]=-gen(2)
↦ [10]:
↦    empty list
↦ [11]:
↦    empty list
~;
↦
↦ -- break point in ./examples/Eresol.sing --
```

D.5.12.3 determinecenter

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: determinecenter(Coef,expJ,c,n,Y,a,listmb,flag,control3,Hhist); Coef, expJ, listmb, flag lists, c number, n, Y, control3 integers, a, Hhist intvec

Compute: next center of blowing up and related information, see example

Return: several lists defining the center and related information

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..4)),dp;
list flag=identifyvar();
ideal J=x(1)^2-x(2)^2*x(3)^5, x(1)*x(3)^3+x(4)^6;
list Lmb=1,list(0,0,0,0),list(0,0,0,0),list(0,0,0,0),iniD(4),iniD(4),list(0,0,0,0),-1
list L=data(J,2,4);
list LL=determinecenter(L[1],L[2],2,4,0,0,Lmb,flag,0,-1); // Compute the first center
LL[1];  // index of variables in the center
↦ [1]:
↦    1
↦ [2]:
↦    4
↦ [3]:
↦    3
↦ [4]:
↦    2
LL[2];  // exponents of ideals J_4,J_3,J_2,J_1
↦ [1]:
↦    [1]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             2
↦          [3]:
↦             5
↦          [4]:
↦             0
↦       [2]:
↦          [1]:
↦             2
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             0
↦    [2]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             6
↦       [2]:
↦          [1]:
↦             1
↦          [2]:
↦             0
↦          [3]:
↦             3
↦          [4]:
↦             0
↦ [2]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 2
↦          [3]:
↦ 5
↦          [4]:
↦ 0
↦    [2]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 0
↦          [4]:
↦ 6
↦    [3]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 6
↦          [4]:
↦ 0
↦ [3]:
```

```
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 2
↦          [3]:
↦ 5
↦          [4]:
↦ 0
↦    [2]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 6
↦          [4]:
↦ 0
↦ [4]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 12
↦          [3]:
↦ 0
↦          [4]:
↦ 0
LL[3];  // list of orders of J_4,J_3,J_2,J_1
↦ [1]:
↦ 2
↦ [2]:
↦ 6
↦ [3]:
↦ 6
↦ [4]:
↦ 12
LL[4];  // list of critical values
↦ [1]:
↦ 2
↦ [2]:
↦ 2
↦ [3]:
↦ 6
↦ [4]:
↦ 6
LL[5];  // components of the resolution function t
↦ [1]:
↦ 1
↦ [2]:
```

```
↦ 3
↦ [3]:
↦ 1
↦ [4]:
↦ 2
LL[6];  // list of D_4,D_3,D_2,D_1
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       0
↦ [2]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       0
↦ [3]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       0
↦ [4]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       0
LL[7];  // list of H_4,H_3,H_2,H_1 (exceptional divisors)
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       0
↦ [2]:
```

```
↦      [1]:
↦         0
↦      [2]:
↦         0
↦      [3]:
↦         0
↦      [4]:
↦         0
↦   [3]:
↦      [1]:
↦         0
↦      [2]:
↦         0
↦      [3]:
↦         0
↦      [4]:
↦         0
↦   [4]:
↦      [1]:
↦         0
↦      [2]:
↦         0
↦      [3]:
↦         0
↦      [4]:
↦         0
LL[8];  // list of all exceptional divisors accumulated
↦   [1]:
↦      0
↦   [2]:
↦      0
↦   [3]:
↦      0
↦   [4]:
↦      0
LL[9];  // auxiliary invariant
↦   [1]:
↦      0
LL[10]; // intvec pointing out the last step where the function t has dropped
↦ -1,-1,-1,-1
ring r= 0,(x(1..4)),dp;
↦ // ** redefining r (ring r= 0,(x(1..4)),dp;) ./examples/determinecenter.s\
   ing:18
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/determinecent\
   er.sing:19
ideal J=x(1)^3-x(2)^2*x(3)^5, x(1)*x(3)^3+x(4)^5;
list Lmb=2,list(0,0,0,0),list(0,0,0,0),list(0,0,0,0),iniD(4),iniD(4),list(0,0,0,0),-
↦ // ** redefining Lmb (=2,list(0,0,0,0),list(0,0,0,0),list(0,0,0,0),iniD(4\
   ),iniD(4),list(0,0,0,0),-1;) ./examples/determinecenter.sing:21
list L2=data(J,2,4);
list L3=determinecenter(L2[1],L2[2],2,4,0,0,Lmb,flag,0,-1); // Example with rational
L3[1]; // index of variables in the center
```

```
↦ [1]:
↦    1
↦ [2]:
↦    3
↦ [3]:
↦    4
L3[2]; // exponents of ideals J_4,J_3,J_2,J_1
↦ [1]:
↦    [1]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             2
↦          [3]:
↦             5
↦          [4]:
↦             0
↦       [2]:
↦          [1]:
↦             3
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             0
↦    [2]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             5
↦       [2]:
↦          [1]:
↦             1
↦          [2]:
↦             0
↦          [3]:
↦             3
↦          [4]:
↦             0
↦ [2]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 2
↦          [3]:
↦ 5
↦          [4]:
↦ 0
↦    [2]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 0
↦          [4]:
↦ 5
↦    [3]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 9/2
↦          [4]:
↦ 0
↦ [3]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 0
↦          [4]:
↦ 5
↦ [4]:
↦    [1]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             0
L3[3]; // list of orders of J_4,J_3,J_2,J_1
↦ [1]:
↦ 3
↦ [2]:
↦ 9/2
↦ [3]:
↦ 5
```

```
L3[4]; // list of critical values
↦ [1]:
↦ 2
↦ [2]:
↦ 3
↦ [3]:
↦ 9/2
L3[5]; // components of the resolution function
↦ [1]:
↦ 3/2
↦ [2]:
↦ 3/2
↦ [3]:
↦ 10/9
```

D.5.12.4 Blowupcenter

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Blowupcenter(center,id,nchart,infochart,c,n,cstep);
center, infochart lists, id, nchart, n, cstep integers, c number

Compute: The blowing up at the chart IDCHART along the given center

Return: new affine charts and related information, see example

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7)),dp;
list flag=identifyvar();
ideal J=x(1)^3-x(3)^2*y(4)^2,x(1)*x(7)*y(2)-x(6)^3*x(5)*y(4)^3,x(5)^3-x(5)^3*y(2)^2;
list Lmb=2,list(0,0,0,0,0,0,0),list(0,0,0,0,0,0,0),list(0,0,0,0,0,0,0),iniD(7),iniD(7
list L=data(J,3,7);
list L2=determinecenter(L[1],L[2],2,7,0,0,Lmb,flag,0,-1); // Computing the center
module auxpath=[0,-1];
list infochart=0,0,0,L[2],L[1],flag,0,list(0,0,0,0,0,0,0),auxpath,list(),list();
list L3=Blowupcenter(L2[1],1,1,infochart,2,7,0);
L3[1]; // current chart (parent,Y,center,expJ,Coef,flag,Hhist,blwhist,path,hipercoef
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦    [4]:
↦       [1]:
↦          [1]:
↦             [1]:
↦                0
↦             [2]:
↦                0
↦             [3]:
↦                2
↦             [4]:
```

```
↦                2
↦             [5]:
↦                0
↦             [6]:
↦                0
↦             [7]:
↦                0
↦          [2]:
↦             [1]:
↦                3
↦             [2]:
↦                0
↦             [3]:
↦                0
↦             [4]:
↦                0
↦             [5]:
↦                0
↦             [6]:
↦                0
↦             [7]:
↦                0
↦       [2]:
↦          [1]:
↦             [1]:
↦                0
↦             [2]:
↦                0
↦             [3]:
↦                0
↦             [4]:
↦                3
↦             [5]:
↦                1
↦             [6]:
↦                3
↦             [7]:
↦                0
↦          [2]:
↦             [1]:
↦                1
↦             [2]:
↦                1
↦             [3]:
↦                0
↦             [4]:
↦                0
↦             [5]:
↦                0
↦             [6]:
↦                0
↦             [7]:
↦                1
↦       [3]:
↦          [1]:
↦             [1]:
↦                0
↦             [2]:
↦                2
↦             [3]:
↦                0
↦             [4]:
↦                0
↦             [5]:
↦                3
↦             [6]:
↦                0
↦             [7]:
↦                0
↦          [2]:
↦             [1]:
↦                0
↦             [2]:
↦                0
↦             [3]:
↦                0
↦             [4]:
↦                0
↦             [5]:
↦                3
↦             [6]:
↦                0
↦             [7]:
↦                0
↦    [5]:
↦       [1]:
↦          [1]:
↦ -1
↦          [2]:
↦ 1
↦       [2]:
↦          [1]:
↦ -1
↦          [2]:
↦ 1
↦       [3]:
↦          [1]:
↦ -1
↦          [2]:
↦ 1
↦    [6]:
↦       [1]:
↦          0
↦       [2]:
↦          1
↦       [3]:
```

```
↦           0
↦         [4]:
↦           1
↦         [5]:
↦           0
↦         [6]:
↦           0
↦         [7]:
↦           0
↦     [7]:
↦       0
↦     [8]:
↦         [1]:
↦           0
↦         [2]:
↦           0
↦         [3]:
↦           0
↦         [4]:
↦           0
↦         [5]:
↦           0
↦         [6]:
↦           0
↦         [7]:
↦           0
↦     [9]:
↦       _[1]=-gen(2)
↦     [10]:
↦       empty list
↦     [11]:
↦       empty list
↦     [12]:
↦       2
↦     [13]:
↦       3
↦     [14]:
↦       4
↦     [15]:
↦       5
↦     [16]:
↦       6
L3[2][1]; // information of its first son, write L3[2][2],...,L3[2][5] to see the oth
↦ [1]:
↦    1
↦ [2]:
↦    3
↦ [3]:
↦    3,1,7,5,6
↦ [4]:
↦    [1]:
↦       [1]:
↦          [1]:
```

```
↦             0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             2
↦          [5]:
↦             0
↦          [6]:
↦             0
↦          [7]:
↦             0
↦       [2]:
↦          [1]:
↦             3
↦          [2]:
↦             0
↦          [3]:
↦ 1
↦          [4]:
↦             0
↦          [5]:
↦             0
↦          [6]:
↦             0
↦          [7]:
↦             0
↦    [2]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦ 2
↦          [4]:
↦             3
↦          [5]:
↦             1
↦          [6]:
↦             3
↦          [7]:
↦             0
↦       [2]:
↦          [1]:
↦             1
↦          [2]:
↦             1
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦             0
↦          [6]:
↦             0
↦          [7]:
↦             1
↦    [3]:
↦       [1]:
↦          [1]:
↦             0
↦          [2]:
↦             2
↦          [3]:
↦ 1
↦          [4]:
↦             0
↦          [5]:
↦             3
↦          [6]:
↦             0
↦          [7]:
↦             0
↦       [2]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦ 1
↦          [4]:
↦             0
↦          [5]:
↦             3
↦          [6]:
↦             0
↦          [7]:
↦             0
↦ [5]:
↦    [1]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦    [2]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦    [3]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦ [6]:
↦    [1]:
↦       0
↦    [2]:
↦       1
↦    [3]:
↦       0
↦    [4]:
↦       1
↦    [5]:
↦       0
↦    [6]:
↦       0
↦    [7]:
↦       0
↦ [7]:
↦    0,3
↦ [8]:
↦    [1]:
↦       0,3
↦    [2]:
↦       0,0
↦    [3]:
↦       0,0
↦    [4]:
↦       0,0
↦    [5]:
↦       0,3
↦    [6]:
↦       0,3
↦    [7]:
↦       0,3
↦ [9]:
↦    _[1]=-gen(2)
↦    _[2]=gen(2)+gen(1)
↦ [10]:
↦    empty list
↦ [11]:
↦    empty list
L3[3];    // current number of charts
↦ 6
L3[4];    // step/level associated to each son
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    1
↦ [5]:
↦    1
L3[5];    // number of variables in the center
```

```
↦ 5
```

D.5.12.5 Nonhyp

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Compute: The "ideal" generated by the non hyperbolic generators of J

Return: lists with the following information
newcoef,newJ: coefficients and exponents of the non hyperbolic generators totalhyp,totalgen: coefficients and exponents of the hyperbolic generators flaglist: new list saying status of variables

Note: the basering r is supposed to be a polynomial ring K[x,y], in fact, we work in a localization of K[x,y], of type K[x,y]_y with y invertible variables.

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7)),dp;
list flag=identifyvar();  // List giving flag=1 to invertible variables: y(2),y(4)
ideal J=x(1)^3-x(3)^2*y(4)^2,x(1)*x(7)*y(2)-x(6)^3*x(5)*y(4)^3,1-x(5)^2*y(2)^2;
list L=data(J,3,7);
list L2=maxEord(L[1],L[2],3,7,flag);
L2[1];     // Maximum E-order
↦ 0
list New=Nonhyp(L[1],L[2],3,7,flag,L2[2]);
New[1];    // Coefficients of the non hyperbolic part
↦ [1]:
↦    [1]:
↦ -1
↦    [2]:
↦ 1
↦ [2]:
↦    [1]:
↦ -1
↦    [2]:
↦ 1
New[2];    // Exponents of the non hyperbolic part
↦ [1]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦          2
↦       [4]:
↦          2
↦       [5]:
↦          0
↦       [6]:
↦          0
↦       [7]:
↦          0
↦    [2]:
```

```
↦       [1]:
↦          3
↦       [2]:
↦          0
↦       [3]:
↦          0
↦       [4]:
↦          0
↦       [5]:
↦          0
↦       [6]:
↦          0
↦       [7]:
↦          0
↦ [2]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦          0
↦       [4]:
↦          3
↦       [5]:
↦          1
↦       [6]:
↦          3
↦       [7]:
↦          0
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦       [3]:
↦          0
↦       [4]:
↦          0
↦       [5]:
↦          0
↦       [6]:
↦          0
↦       [7]:
↦          1
New[3];    // Coefficients of the hyperbolic part
↦ [1]:
↦    [1]:
↦ -1
↦    [2]:
↦ 1
New[4];    // New hyperbolic equations
↦ [1]:
```

```
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          2
↦       [3]:
↦          0
↦       [4]:
↦          0
↦       [5]:
↦          2
↦       [6]:
↦          0
↦       [7]:
↦          0
↦    [2]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦          0
↦       [4]:
↦          0
↦       [5]:
↦          0
↦       [6]:
↦          0
↦       [7]:
↦          0
New[5];    // New list giving flag=1 to invertible variables: y(2),y(4),y(5)
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    0
↦ [4]:
↦    1
↦ [5]:
↦    1
↦ [6]:
↦    0
↦ [7]:
↦    0
ring r = 0,(x(1..4)),dp;
↦ // ** redefining r (ring r = 0,(x(1..4)),dp;) ./examples/Nonhyp.sing:14
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/Nonhyp.sing:1\
   5
ideal J=1-x(1)^5*x(2)^2*x(3)^5, x(1)^2*x(3)^3+x(1)^4*x(4)^6;
list L=data(J,2,4);
list L2=maxEord(L[1],L[2],2,4,flag);
```

```
L2[1];     // Maximum E-order
↦ 0
list New=Nonhyp(L[1],L[2],2,4,flag,L2[2]);
New;
↦ [1]:
↦    empty list
↦ [2]:
↦    empty list
↦ [3]:
↦    [1]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦    [2]:
↦       [1]:
↦ 1
↦       [2]:
↦ 1
↦ [4]:
↦    [1]:
↦       [1]:
↦          [1]:
↦             5
↦          [2]:
↦             2
↦          [3]:
↦             5
↦          [4]:
↦             0
↦       [2]:
↦          [1]:
↦             0
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             0
↦    [2]:
↦       [1]:
↦          [1]:
↦             4
↦          [2]:
↦             0
↦          [3]:
↦             0
↦          [4]:
↦             6
↦       [2]:
↦          [1]:
↦             2
↦          [2]:
```

```
↦              0
↦           [3]:
↦              3
↦           [4]:
↦              0
↦    [5]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦       [3]:
↦          1
↦       [4]:
↦          1
```

D.5.12.6 identifyvar

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: identifyvar();

Compute: Assign 0 to variables x and 1 to variables y, only necessary at the beginning

Return: list, say l, of size the dimension of the basering
l[i] is: 0 if the i-th variable is x(i),
1 if the i-th variable is y(i)

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
identifyvar();
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    0
↦ [4]:
↦    1
↦ [5]:
↦    0
↦ [6]:
↦    0
↦ [7]:
↦    0
↦ [8]:
↦    1
```

D.5.12.7 Edatalist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Edatalist(Coef,Exp,k,n,flaglist);
Coef,Exp,flaglist lists, k,n, integers
Exp is a list of lists of exponents, k=size(Exp)

Compute: computes a list with the E-order of each term

Return: a list with the E-order of each term

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
list flag=identifyvar();
ideal J=x(1)^3*x(3)-y(2)*y(4)^2,x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5);
list L=data(J,3,8);
list EL=Edatalist(L[1],L[2],3,8,flag);
EL; // E-order of each term
↦ [1]:
↦    [1]:
↦ 4
↦    [2]:
↦ 0
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦ [3]:
↦    [1]:
↦ 2
↦    [2]:
↦ 2
ring r = 2,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
↦ // ** redefining r (ring r = 2,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;) ./\
   examples/Edatalist.sing:8
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/Edatalist.sin\
   g:9
ideal J=x(1)^3*x(3)-y(2)*y(4)^2,x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5);
list L=data(J,3,8);
list EL=Edatalist(L[1],L[2],3,8,flag);
EL; // E-order of each term IN CHAR 2, COMPUTATIONS NEED TO BE DONE IN CHAR 0
↦ [1]:
↦    [1]:
↦ 0
↦    [2]:
↦ 0
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ 1
↦ [3]:
↦    [1]:
↦ 0
↦    [2]:
↦ 0
ring r = 0,(x(1..3)),dp;
↦ // ** redefining r (ring r = 0,(x(1..3)),dp;) ./examples/Edatalist.sing:1\
   4
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/Edatalist.sin\
   g:15
ideal J=x(1)^4*x(2)^2, x(1)^2-x(3)^3;
list L=data(J,2,3);
list EL=Edatalist(L[1],L[2],2,3,flag);
EL; // E-order of each term
↦ [1]:
↦    [1]:
↦ 6
↦ [2]:
↦    [1]:
↦ 3
↦    [2]:
↦ 2
```

D.5.12.8 EOrdlist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: EOrdlist(Coef,Exp,k,n,flaglist);
Coef,Exp,flaglist lists, k,n, integers
Exp is a list of lists of exponents, k=size(Exp)

Compute: computes de E-order of an ideal given by a list (Coef,Exp) and extra information

Return: maximal E-order, and its position=number of generator and term

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
list flag=identifyvar();
ideal J=x(1)^3*x(3)-y(2)*y(4)^2,x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5),x(7)^4*y
list L=data(J,4,8);
list Eo=EOrdlist(L[1],L[2],4,8,flag);
Eo[1]; // E-order
↦ 0
Eo[2]; // generator giving the E-order
↦ 1
Eo[3]; // term giving the E-order
↦ 2
```

D.5.12.9 maxEord

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: maxEord(Coef,Exp,k,n,flaglist);
Coef,Exp,flaglist lists, k,n, integers
Exp is a list of lists of exponents, k=size(Exp)

Return: computes de maximal E-order of an ideal given by Coef,Exp

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
```

```
list flag=identifyvar();
ideal J=x(1)^3*x(3)-y(2)*y(4)^2*x(3),x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5),x(7
list L=data(J,4,8);
list M=maxEord(L[1],L[2],4,8,flag);
M[1];    // E-order
↦ 1
```

D.5.12.10 ECoef

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: ECoef(Coef,expP,sP,V,auxc,n,flaglist);
Coef, expP, flaglist lists, sP, V, n integers, auxc number

Compute: The ideal E-Coeff_V(P), where V is a permissible hypersurface which belongs to the center

Return: list of exponents, list of coefficients and classification of the ideal E-Coeff_V(P)

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7)),dp;
list flag=identifyvar();
ideal P=x(1)^2*x(3)^5-x(5)^7*y(4),x(6)^3*y(2)^5-x(7)^5,x(5)^3*x(6)-y(4)^3*x(1)^5;
list L=data(P,3,7);
list L2=ECoef(L[1],L[2],3,1,3,7,flag);
L2[1];  // exponents of the E-Coefficient ideal respect to x(1)
↦ [1]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦          0
↦       [3]:
↦ 0
↦       [4]:
↦          0
↦       [5]:
↦ 7
↦       [6]:
↦ 0
↦       [7]:
↦ 0
↦ [2]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦          0
↦       [3]:
↦ 15
↦       [4]:
↦          0
↦       [5]:
↦ 0
↦       [6]:
↦ 0
↦       [7]:
↦ 0
↦ [3]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦ 5
↦       [3]:
↦ 0
↦       [4]:
↦ 0
↦       [5]:
↦ 0
↦       [6]:
↦ 3
↦       [7]:
↦ 0
↦    [2]:
↦       [1]:
↦ 0
↦       [2]:
↦ 0
↦       [3]:
↦ 0
↦       [4]:
↦ 0
↦       [5]:
↦ 0
↦       [6]:
↦ 0
↦       [7]:
↦ 5
↦ [4]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦          0
↦       [3]:
↦ 0
↦       [4]:
↦          0
↦       [5]:
↦ 3
↦       [6]:
↦ 1
↦       [7]:
↦ 0
L2[2];  // its coefficients
↦ [1]:
```

```
↦ -1
↦ [2]:
↦ 1
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦ -1
↦ [4]:
↦ 1
L2[3];  // classify the type of ideal obtained
↦ 0
ring r = 0,(x(1),y(2),x(3),y(4)),dp;
↦ // ** redefining r (ring r = 0,(x(1),y(2),x(3),y(4)),dp;) ./examples/ECoe\
   f.sing:10
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/ECoef.sing:11
ideal J=x(1)^3*(1-2*y(2)*y(4)^2);  // Bold regular case
list L=data(J,1,4);
list L2=ECoef(L[1],L[2],1,1,3,4,flag);
L2;
↦ [1]:
↦    empty list
↦ [2]:
↦    empty list
↦ [3]:
↦    1
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7)),dp;
↦ // ** redefining r (ring r = 0,(x(1),y(2),x(3),y(4),x(5..7)),dp;) ./examp\
   les/ECoef.sing:16
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/ECoef.sing:17
ideal J=x(1)^3-x(3)^2*y(4)^2,x(1)*x(7)*y(2)-x(6)^3*x(5)*y(4)^3,x(5)^3-x(5)^3*y(2)^2;
list L=data(J,3,7);
list L2=ECoef(L[1],L[2],3,1,2,7,flag);
↦ // ** redefining L2 (list L2=ECoef(L[1],L[2],3,1,2,7,flag);) ./examples/E\
   Coef.sing:20
L2;
↦ [1]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 2
↦          [4]:
↦             0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦    [2]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦ 1
↦          [6]:
↦ 3
↦          [7]:
↦ 0
↦    [3]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 2
↦    [4]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦ 2
↦          [3]:
↦ 0
↦          [4]:
↦ 0
↦          [5]:
↦ 3
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦       [2]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 0
↦          [4]:
↦ 0
↦          [5]:
↦ 3
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦ [2]:
↦    [1]:
↦ -1
↦    [2]:
↦ -1
↦    [3]:
↦ 1
↦    [4]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦ [3]:
↦    0
ring r = 3,(x(1),y(2),x(3),y(4),x(5..7)),dp;
↦ // ** redefining r (ring r = 3,(x(1),y(2),x(3),y(4),x(5..7)),dp;) ./examp\
  les/ECoef.sing:22
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/ECoef.sing:23
ideal J=x(1)^3-x(3)^2*y(4)^2,x(1)*x(7)*y(2)-x(6)^3*x(5)*y(4)^3,x(5)^3-x(5)^3*y(2)^2;
list L=data(J,3,7);
list L2=ECoef(L[1],L[2],3,1,2,7,flag);
↦ E-order zero!
L2; // THE COMPUTATIONS ARE NOT CORRECT IN CHARACTERISTIC p>0
↦ [1]:
↦    [1]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ -1
↦          [4]:
↦             0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦    [2]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦    [3]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦ 1
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦    [4]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
↦             0
↦          [3]:
↦ 0
↦          [4]:
↦             0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ -1
↦    [5]:
↦       [1]:
↦          [1]:
↦ 0
↦          [2]:
```

```
↦ -1
↦          [3]:
↦ 0
↦          [4]:
↦ 0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦       [2]:
↦          [1]:
↦ 0
↦          [2]:
↦ 0
↦          [3]:
↦ 0
↦          [4]:
↦ 0
↦          [5]:
↦ 0
↦          [6]:
↦ 0
↦          [7]:
↦ 0
↦ [2]:
↦    [1]:
↦ -1
↦    [2]:
↦ 1
↦    [3]:
↦ -1
↦    [4]:
↦ 1
↦    [5]:
↦       [1]:
↦ -1
↦       [2]:
↦ 1
↦ [3]:
↦    2
// because numbers are treated as 0 in assignments
```

D.5.12.11 elimrep

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: elimrep(L); L is a list

Compute: Eliminate repeated terms from a list

Return: the same list without repeated terms

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list L=4,5,2,5,7,8,6,3,2;
elimrep(L);
↦ [1]:
↦    4
↦ [2]:
↦    5
↦ [3]:
↦    2
↦ [4]:
↦    7
↦ [5]:
↦    8
↦ [6]:
↦    6
↦ [7]:
↦    3
```

D.5.12.12 Emaxcont

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Emaxcont(Coef,Exp,k,n,flag);
Coef,Exp,flag lists, k,n, integers
Exp is a list of lists of exponents, k=size(Exp)

Compute: Identify ALL the variables of E-maximal contact

Return: a list with the indexes of the variables of E-maximal contact

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
list flag=identifyvar();
ideal J=x(1)^3*x(3)-y(2)*y(4)^2,x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5),x(7)^4*y
list L=data(J,4,8);
list hyp=Emaxcont(L[1],L[2],4,8,flag);
hyp[1]; // max E-order=0
↦ 0
hyp[2]; // There are no hypersurfaces of E-maximal contact
↦ empty list
ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;
↦ // ** redefining r (ring r = 0,(x(1),y(2),x(3),y(4),x(5..7),y(8)),dp;) ./\
   examples/Emaxcont.sing:9
list flag=identifyvar();
↦ // ** redefining flag (list flag=identifyvar();) ./examples/Emaxcont.sing\
   :10
ideal J=x(1)^3*x(3)-y(2)*y(4)^2*x(3),x(5)*y(2)-x(7)*y(4)^2,x(6)^2*(1-y(4)*y(8)^5),x(7
list L=data(J,4,8);
list hyp=Emaxcont(L[1],L[2],4,8,flag);
hyp[1]; // the E-order is 1
↦ 1
hyp[2]; // {x(3)=0},{x(5)=0},{x(7)=0} are hypersurfaces of E-maximal contact
↦ [1]:
```

```
↦    3
↦ [2]:
↦    7
↦ [3]:
↦    5
```

D.5.12.13 cleanunit

Procedure from library resbinomial.lib (see Section D.5.12 [resbinomial_lib], page 1475).

Compute: We clean (or forget) the units in a monomial, given by "y" variables

Return: The list defining the monomial ideal already cleaned

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1),y(2),x(3),y(4)),dp;
list flag=identifyvar();
ideal J=x(1)^3*y(2)*x(3)^5*y(4)^8;
list L=data(J,1,4);
L[2][1][1];  // list of exponents of the monomial J
↦ [1]:
↦    3
↦ [2]:
↦    1
↦ [3]:
↦    5
↦ [4]:
↦    8
list M=cleanunit(L[2][1][1],4,flag);
M;            // new list without units
↦ [1]:
↦    3
↦ [2]:
↦    0
↦ [3]:
↦    5
↦ [4]:
↦    0
```

D.5.12.14 resfunction

Procedure from library resbinomial.lib (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: resfunction(invariant,auxinv,nchart,n);
invariant, auxinv lists, nchart, n integers

Compute: Patch the resolution function

Return: The complete resolution function

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^2-x(2)^3;
list L=Eresol(J);
```

```
L[3]; // incomplete resolution function
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ 3/2
↦ [2]:
↦    @
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦ 2
↦ [4]:
↦    @
↦ [5]:
↦    [1]:
↦ 1
↦    [2]:
↦ 0
↦ [6]:
↦    #
↦ [7]:
↦    #
resfunction(L[3],L[7],7,2); // complete resolution function
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ 3/2
↦ [2]:
↦    @
↦ [3]:
↦    [1]:
↦ 1
↦    [2]:
↦ 2
↦ [4]:
↦    @
↦ [5]:
↦    [1]:
↦ 1
↦    [2]:
↦       [1]:
↦          -1
↦       [2]:
↦ 1
↦       [3]:
↦          1
↦ [6]:
↦    #
↦ [7]:
↦    #
```

D.5.12.15 calculateI

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: calculateI(Coef,expJ,c,n,Y,a,b,D);
Coef, expJ, D lists, c, b numbers, n,Y integers, a intvec

Return: ideal I, non monomial part of J

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list flag=identifyvar();
ideal J=x(1)^4*x(2)^2, x(3)^3;
list Lmb=1,list(0,0,0),list(0,0,0),list(3),iniD(3),iniD(3),list(0,0,0),-1;
list L=data(J,2,3);
list LL=determinecenter(L[1],L[2],3,3,0,0,Lmb,flag,0,-1); // Calculate the center
module auxpath=[0,-1];
list infochart=0,0,0,L[2],L[1],flag,0,list(0,0,0),auxpath,list(),list();
list L3=Blowupcenter(LL[1],1,1,infochart,3,3,0); // blowing-up and looking to the x(
calculateI(L3[2][1][5],L3[2][1][4],3,3,3,L3[2][1][3],3,iniD(3)); //  (I_3)
↦ [1]:
↦    [1]:
↦       [1]:
↦          4
↦       [2]:
↦          2
↦       [3]:
↦ 3
↦ [2]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦ 0
// looking to the x(1) chart
calculateI(L3[2][2][5],L3[2][2][4],3,3,1,L3[2][2][3],3,iniD(3)); //  (I_3)
↦ [1]:
↦    [1]:
↦       [1]:
↦ 3
↦       [2]:
↦          2
↦       [3]:
↦          0
↦ [2]:
↦    [1]:
↦       [1]:
↦ 0
↦       [2]:
↦          0
↦       [3]:
↦          3
```

D.5.12.16 Maxord

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Maxord(L,n); L list, n integer

Compute: Find the maximal entry of a list, input is a list defining a monomial

Return: maximum entry of a list and its position

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
ideal J=x(1)^2*x(2)*x(3)^5;
list L=data(J,1,3);
L[2]; // list of exponents
↦ [1]:
↦    [1]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦       [3]:
↦          5
Maxord(L[2][1][1],3);
↦ 5 3
```

D.5.12.17 Gamma

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: Gamma(L,c,n); L list, c number, n integer

Compute: The Gamma function, resolution function corresponding to the monomial case

Return: lists of maximum exponents in L, value of Gamma function, center of blow up

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..5)),dp;
ideal J=x(1)^2*x(2)*x(3)^5*x(4)^2*x(5)^3;
list L=data(J,1,5);
list G=Gamma(L[2][1][1],9,5);   // critical value c=9
G[1]; // maximum exponents in the ideal
↦ [1]:
↦ 5
↦ [2]:
↦ 3
↦ [3]:
↦ 2
G[2]; // maximal value of Gamma function
↦ [1]:
↦    -3
↦ [2]:
↦ 10/9
↦ [3]:
↦    3,5,4
```

```
G[3]; // center given by Gamma
↦ [1]:
↦    3
↦ [2]:
↦    5
↦ [3]:
↦    4
```

D.5.12.18 convertdata

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: convertdata(C,L,n,flaglist);
C, L, flaglist lists, n integer

Compute: Compute the ideal corresponding to the given lists C,L

Return: an ideal whose coefficients are given by C, exponents given by L

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..4),y(5)),dp;
list M=identifyvar();
ideal J=x(1)^2*y(5)^2-x(2)^2*x(3)^2,6*x(4)^2;
list L=data(J,2,5);
L[1]; // Coefficients
↦ [1]:
↦    [1]:
↦ -1
↦    [2]:
↦ 1
↦ [2]:
↦    [1]:
↦ 6
L[2]; // Exponents
↦ [1]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          2
↦       [3]:
↦          2
↦       [4]:
↦          0
↦       [5]:
↦          0
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          0
↦       [3]:
↦          0
↦       [4]:
```

```
↦          0
↦       [5]:
↦          2
↦ [2]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦          0
↦       [4]:
↦          2
↦       [5]:
↦          0
ideal J2=convertdata(L[1],L[2],5,M);
J2;
↦ J2[1]=-x(2)^2*x(3)^2+x(1)^2*y(5)^2
↦ J2[2]=6*x(4)^2
```

D.5.12.19 lcmofall

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: lcmofall(nchart,mobile);
nchart integer, mobile list of lists

Compute: Compute the lcm of the denominators of the E-orders of all the charts

Return: an integer given the lcm

Note: CALL BEFORE salida

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^3-x(1)*x(2)^3;
list L=Eresol(J);
L[4];  // 8 charts, rational exponents
↦ 8
L[8][2][2]; // E-orders at the first chart
↦ [1]:
↦ 3
↦ [2]:
↦ 9/2
lcmofall(8,L[8]);
↦ 2
```

D.5.12.20 computemcm

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: computemcm(Eolist); Eolist list

Return: an integer, the least common multiple of the denominators of the E-orders

Note: Make the same as lcmofall but for one chart. NECESSARY BECAUSE THE E-ORDERS ARE OF TYPE NUMBER!!

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^3-x(1)*x(2)^3;
list L=Eresol(J);   // 8 charts, rational exponents
L[8][2][2];         // maximal E-order at the first chart
↦ [1]:
↦ 3
↦ [2]:
↦ 9/2
computemcm(L[8][2][2]);
↦ 2
```

D.5.12.21 constructH

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: constructH(Hhist,n,flag);
Hhist intvec, n integer, flag list

Return: the list of exceptional divisors accumulated at this chart

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list flag=identifyvar();
ideal J=x(1)^4*x(2)^2, x(1)^2+x(3)^3;
list L=Eresol(J);   // 7 charts
// history of the exceptional divisors at the 7-th chart
L[1][7][7]; // blow ups at x(3)-th, x(1)-th and x(1)-th charts
↦ 0,3,1,1
constructH(L[1][7][7],3,flag);
↦ [1]:
↦    _[1]=x(3)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)
```

D.5.12.22 constructblwup

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: constructblwup(blwhist,n,chy,flag);
blwhist, flag lists, n integer, chy ideal

Return: the ideal defining the map K[W] –> K[Wi],
which gives the composition map of all the blowing up leading to this chart

Note: NECESSARY START WITH COLUMNS

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list flag=identifyvar();
ideal chy=maxideal(1);
ideal J=x(1)^4*x(2)^2, x(1)^2+x(3)^3;
list L=Eresol(J);   // 7 charts
// history of the blow ups at the 7-th chart, center {x(1)=x(3)=0} every time
L[1][7][8]; // blow ups at x(3)-th, x(1)-th and x(1)-th charts
↦ [1]:
↦    0,3,0,0
↦ [2]:
↦    0,0,0,0
↦ [3]:
↦    0,0,1,1
constructblwup(L[1][7][8],3,chy,flag);
↦ _[1]=x(1)^3*x(3)
↦ _[2]=x(2)
↦ _[3]=x(1)^2*x(3)
```

D.5.12.23 constructlastblwup

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: constructlastblwup(blwhist,n,chy,flag);
blwhist, flag lists, n integer, chy ideal

Return: the ideal defining the last blow up

Note: NECESSARY START WITH COLUMNS

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list flag=identifyvar();
ideal chy=maxideal(1);
ideal J=x(1)^4*x(2)^2, x(1)^2+x(3)^3;
list L=Eresol(J);   // 7 charts
// history of the blow ups at the 7-th chart, center {x(1)=x(3)=0} every time
L[1][7][8]; // blow ups at x(3)-th, x(1)-th and x(1)-th charts
↦ [1]:
↦    0,3,0,0
↦ [2]:
↦    0,0,0,0
↦ [3]:
↦    0,0,1,1
constructlastblwup(L[1][7][8],3,chy,flag);
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ _[3]=x(1)*x(3)
```

D.5.12.24 genoutput

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: genoutput(chart,mobile,nchart,nsons,n,q,p);
chart, mobile, nsons lists, nchart, n,q, p integers

Return: two lists, the first one gives the rings corresponding to the final charts, the second one is the list of all rings corresponding to the affine charts of the resolution process

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^3-x(1)*x(2)^3;
list L=Eresol(J);         // 8 charts, rational exponents
list B=genoutput(L[1],L[8],L[4],L[6],2,2,0); // generates the output
presentTree(B);
↦
↦ /////////////////////////// Final Chart 1 /////////////////////////
↦ ======================== History of this chart ======================
↦
↦ Blow Up 1 :
↦      Center determined in L[2][1],
↦      Passing to chart  1  in resulting blow up.
↦
↦ ======================== Data of this chart ========================
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=-y(1)*y(2)^3+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(1)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=y(1)
↦ _[2]=y(1)*y(2)
↦
↦ pause>
↦ /////////////////////////// Final Chart 2 /////////////////////////
↦ ======================== History of this chart ======================
↦
↦ Blow Up 1 :
↦      Center determined in L[2][1],
↦      Passing to chart  2  in resulting blow up.
↦
↦ Blow Up 2 :
↦      Center determined in L[2][3],
↦      Passing to chart  1  in resulting blow up.
↦
↦ Blow Up 3 :
↦      Center determined in L[2][4],
↦      Passing to chart  1  in resulting blow up.
↦
↦ ======================== Data of this chart ========================
↦
↦ ==== Ambient Space:
```

```
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=y(1)-1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(2)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=y(1)^2*x(2)^3
↦ _[2]=y(1)*x(2)^2
↦
↦ pause>
↦ /////////////////////////// Final Chart 3 /////////////////////////
↦ ======================== History of this chart ======================
↦
↦ Blow Up 1 :
↦      Center determined in L[2][1],
↦      Passing to chart  2  in resulting blow up.
↦
↦ Blow Up 2 :
↦      Center determined in L[2][3],
↦      Passing to chart  1  in resulting blow up.
↦
↦ Blow Up 3 :
↦      Center determined in L[2][4],
↦      Passing to chart  2  in resulting blow up.
↦
↦ ======================== Data of this chart ========================
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=-y(2)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(2)
↦ _[2]=x(1)^2*y(2)
↦
```

```
↦ pause>
↦ /////////////////////////// Final Chart 4 /////////////////////////
↦ ======================== History of this chart ======================
↦ 
↦ Blow Up 1 :
↦      Center determined in L[2][1],
↦      Passing to chart  2  in resulting blow up.
↦ 
↦ Blow Up 2 :
↦      Center determined in L[2][3],
↦      Passing to chart  2  in resulting blow up.
↦ 
↦ Blow Up 3 :
↦      Center determined in L[2][5],
↦      Passing to chart  1  in resulting blow up.
↦ 
↦ ======================== Data of this chart ========================
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=y(1)^2*y(2)-1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(2)
↦ [3]:
↦    _[1]=y(1)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=y(1)*y(2)^2
↦ _[2]=y(2)
↦ 
↦ pause>///////////////////////////////////////////////////////////////////\
   /
↦ For identification of exceptional divisors please use the tools
↦ provided by reszeta.lib, e.g. collectDiv.
↦ For viewing an illustration of the tree of charts please use the
↦ procedure ResTree from resgraph.lib.
↦ ////////////////////////////////////////////////////////////////////
list iden0=collectDiv(B);
ResTree(B,iden0[1]);        // generates the resolution tree
↦ sh: dot: Kommando nicht gefunden.
↦ Currently showing graphics in separate window
↦ Press <Return> to continue
↦ sh: display: Kommando nicht gefunden.
↦ pause>./examples/genoutput.sing   9> // Use presentTree(B); to see the fi\
   nal charts
// To see the tree type in another shell
//    dot -Tjpg ResTree.dot -o ResTree.jpg
//   /usr/bin/X11/xv ResTree.jpg
```

D.5.12.25 salida

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: salida(idchart,chart,mobile,numson,previousa,n,q,p);
idchart, numson, n, q, p integers, chart, mobile, lists, previousa intvec

Compute: CONVERT THE OUTPUT OF A CHART IN A RING, WHERE DEFINE A BASIC OBJECT (BO)

Return: the ring corresponding to the chart

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..2)),dp;
ideal J=x(1)^2-x(2)^3;
list L=Eresol(J);
list B=salida(5,L[1][5],L[8][6],2,L[1][3][3],2,1,0); // chart 5
def RR=B[1];
setring RR;
BO;
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    _[1]=x(1)-x(2)
↦ [3]:
↦    1,0
↦ [4]:
↦    [1]:
↦       _[1]=x(2)
↦    [2]:
↦       _[1]=x(1)
↦ [5]:
↦    _[1]=x(1)^2*x(2)
↦    _[2]=x(1)*x(2)
↦ [6]:
↦    0,0
↦ [7]:
↦    2,-1
↦ [8]:
↦    _[1,1]=0
↦    _[1,2]=1
↦    _[2,1]=0
↦    _[2,2]=0
↦ [9]:
↦    0,0
"press return to see next example"; ~;
↦ press return to see next example
↦ 
↦ -- break point in ./examples/salida.sing --
```

D.5.12.26 iniD

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: iniD(n); n integer

Return: list of lists of zeros of size n

Example:

```
LIB "resbinomial.lib";
iniD(3);
↦ [1]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦ [2]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
↦ [3]:
↦    [1]:
↦       0
↦    [2]:
↦       0
↦    [3]:
↦       0
```

D.5.12.27 sumlist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: sumlist(L1,L2); L1,L2 lists, (size(L1)==size(L2))

Return: a list, sum of L1 and L2

Example:

```
LIB "resbinomial.lib";
list L1=1,2,3;
list L2=5,9,7;
sumlist(L1,L2);
↦ [1]:
↦    6
↦ [2]:
↦    11
↦ [3]:
↦    10
```

D.5.12.28 reslist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: reslist(L1,L2); L1,L2 lists, (size(L1)==size(L2))

Return: a list, subtraction of L1 and L2

Example:

```
LIB "resbinomial.lib";
list L1=1,2,3;
list L2=5,9,7;
reslist(L1,L2);
↦ [1]:
↦    -4
↦ [2]:
↦    -7
↦ [3]:
↦    -4
```

D.5.12.29 multiplylist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: multiplylist(L,a); L list, a number

Return: list of elements of type number, multiplication of L times a

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list L=1,2,3;
multiplylist(L,1/5);
↦ [1]:
↦ 1/5
↦ [2]:
↦ 2/5
↦ [3]:
↦ 3/5
```

D.5.12.30 dividelist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: dividelist(L1,L2); L1,L2 lists

Return: list of elements of type number, division of L1 by L2

Example:

```
LIB "resbinomial.lib";
ring r = 0,(x(1..3)),dp;
list L1=1,2,3;
list L2=5,9,7;
dividelist(L1,L2);
↦ [1]:
↦ 1/5
↦ [2]:
↦ 2/9
↦ [3]:
↦ 3/7
```

D.5.12.31 createlist

Procedure from library `resbinomial.lib` (see Section D.5.12 [resbinomial_lib], page 1475).

Usage: createlist(L1,L2); L1,L2 lists, (size(L1)==size(L2))

Return: list of lists of two elements, the first one of L1 and the second of L2

Example:

```
LIB "resbinomial.lib";
list L1=1,2,3;
list L2=5,9,7;
createlist(L1,L2);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       5
↦ [2]:
↦    [1]:
↦       2
↦    [2]:
↦       9
↦ [3]:
↦    [1]:
↦       3
↦    [2]:
↦       7
```

D.5.13 resgraph_lib

Library: resgraph.lib

Purpose: Visualization of Resolution Data

Author: A. Fruehbis-Krueger, anne@mathematik.uni-kl.de,

Note: This library uses the external programs surf, graphviz and imagemagick.
Input data is assumed to originate from resolve.lib and reszeta.lib

Procedures:

D.5.13.1 InterDiv

Procedure from library `resgraph.lib` (see Section D.5.13 [resgraph_lib], page 1523).

Usage: InterDiv(M[,name]);
M = matrix
name = string

Assume: - M is first list entry of output of 'intersectionDiv'
from library reszeta.lib
- write permission in the current directory or in the
directory in which the file with name 'name' resides

Create: file 'name.jpg' containing dual graph of resolution
if filename is given

Note: only available on UNIX-type systems and programs
'display' (imagemagick package) and 'dot' (Graphviz package) need to
be in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.5.13.2 ResTree

Procedure from library `resgraph.lib` (see Section D.5.13 [resgraph_lib], page 1523).

Usage: ResTree(L,M[,name][,mark]);
L = list
M = matrix
name = string
mark = intvec

Assume: - L is the output of 'resolve' from resolve.lib
- M is first entry of output of 'collectDiv(L);' from reszeta.lib
- write permission in the current directory or in the
directory in which the file with name 'name' resides
- mark intvec of size size(L[2])
mark[i]=0 (default) border of box black
mark[i]=1 border of box red

Create: file 'name.jpg' containing the tree of charts of L
if filename is given

Note: only available on UNIX-type systems and programs
'display' (imagemagick package) and 'dot' (Graphviz package) need to
be in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.5.13.3 finalCharts

Procedure from library `resgraph.lib` (see Section D.5.13 [resgraph_lib], page 1523).

Usage: finalCharts(L1,L2,iv[,name]);
L1 = list
L2 = list
iv = intvec
name = string

Assume: - L1 is the output of 'resolve' from resolve.lib
- L2 is the output of 'intersectionDiv(L1)' from reszeta.lib
- iv is the first entry of the output of 'abstractR(L1)'
- write permission in the current directory or in the
directory in which the file with name 'name' resides

Create: - new windows in which surf-images of the final charts are presented
- several '.ras' files in the directory in which 'name' resides

Note: only available on UNIX-type systems
external programs 'surf' and 'display' (imagemagick package) need to be
in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.5.14 resjung_lib

Library: resjung.lib

Purpose: Resolution of surface singularities (Desingularization) Algorithm of Jung

Author: Philipp Renner, philipp_renner@web.de

Procedures:

D.5.14.1 jungresolve

Procedure from library `resjung.lib` (see Section D.5.14 [resjung_lib], page 1525).

Usage: jungresolve(ideal J[,is_noeth]);
J = ideal
i = int

Assume: J = two dimensional ideal

Return: a list l of rings
l[i] is a ring containing two Ideals: QIdeal and BMap. BMap defines a birational morphism from V(QIdeal)–>V(J), such that V(QIdeal) is smooth. For this the algorithm computes first with jungnormal a representation of V(J) with Hirzebruch-Jung singularities and then it uses Villamayor's algorithm to resolve these singularities If is_noeth=1 the algorithm assumes J is in noether position with respect to the last two variables. As a default or if is_noeth = 0 the algorithm computes a coordinate change such that J is in noether position. NOTE: since the noether position algorithm is randomized the performance can vary significantly.

Example:

```
LIB "resjung.lib";
//Computing a resolution of singularities of the variety z2-x3-y3
ring r = 0,(x,y,z),dp;
ideal I = z2-x3-y3;
//The ideal is in noether position
list l = jungresolve(I,1);
def R1 = l[1];
def R2 = l[2];
setring R1;
QIdeal;
↦ QIdeal[1]=T(1)*x(2)^2*y(1)-T(2)*x
↦ QIdeal[2]=T(2)*x(2)^2*y(1)-x
↦ QIdeal[3]=x(2)^2*y(1)^2-x(2)^2*y(1)-T(1)*x
↦ QIdeal[4]=T(1)^2-T(2)*y(1)+T(2)
↦ QIdeal[5]=T(1)*T(2)-y(1)+1
↦ QIdeal[6]=T(2)^2-T(1)
↦ QIdeal[7]=x(2)^6*y(1)^4-x(2)^6*y(1)^3-x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(2)^2*y(1)
↦ BMap[3]=x(2)^3*y(1)^2
setring R2;
QIdeal;
↦ QIdeal[1]=T(2)*x(1)^2+x
↦ QIdeal[2]=T(1)*x(1)^2*y(0)-T(2)*x
↦ QIdeal[3]=x(1)^2*y(0)^2-x(1)^2*y(0)-T(1)*x
↦ QIdeal[4]=T(1)^2-T(2)*y(0)+T(2)
↦ QIdeal[5]=T(1)*T(2)+y(0)^2-y(0)
↦ QIdeal[6]=T(2)^2+T(1)*y(0)
↦ QIdeal[7]=x(1)^6*y(0)^3-x(1)^6*y(0)^2+x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(1)^2*y(0)
↦ BMap[3]=x(1)^3*y(0)
```

D.5.14.2 jungnormal

Procedure from library `resjung.lib` (see Section D.5.14 [resjung_lib], page 1525).

Usage: jungnormal(ideal J[,is_noeth]);
J = ideal
i = int

Assume: J = two dimensional ideal

Return: a list l of rings
l[i] is a ring containing two Ideals: QIdeal and BMap. BMap defines a birational morphism from V(QIdeal)–>V(J), such that V(QIdeal) has only singularities of Hizebuch-Jung type. If is_noeth=1 the algorithm assumes J is in noether position with respect to the last two variables. As a default or if is_noeth = 0 the algorithm computes a coordinate change such that J is in noether position. NOTE: since the noether position algorithm is randomized the performance can vary significantly.

Example:

```
LIB "resjung.lib";
//Computing a resolution of singularities of the variety z2-x3-y3
ring r = 0,(x,y,z),dp;
ideal I = z2-x3-y3;
//The ideal is in noether position
list l = jungnormal(I,1);
def R1 = l[1];
def R2 = l[2];
setring R1;
QIdeal;
↦ QIdeal[1]=T(1)*x(2)^2*y(1)-T(2)*x
↦ QIdeal[2]=T(2)*x(2)^2*y(1)-x
↦ QIdeal[3]=-T(1)*x+x(2)^2*y(1)^2-x(2)^2*y(1)
↦ QIdeal[4]=T(1)^2-T(2)*y(1)+T(2)
↦ QIdeal[5]=T(1)*T(2)-y(1)+1
↦ QIdeal[6]=T(2)^2-T(1)
↦ QIdeal[7]=x(2)^6*y(1)^4-x(2)^6*y(1)^3-x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(2)^2*y(1)
↦ BMap[3]=x(2)^3*y(1)^2
setring R2;
QIdeal;
↦ QIdeal[1]=T(2)*x(1)^2+x
↦ QIdeal[2]=T(1)*x(1)^2*y(0)-T(2)*x
↦ QIdeal[3]=-T(1)*x+x(1)^2*y(0)^2-x(1)^2*y(0)
↦ QIdeal[4]=T(1)^2-T(2)*y(0)+T(2)
↦ QIdeal[5]=T(1)*T(2)+y(0)^2-y(0)
↦ QIdeal[6]=T(2)^2+T(1)*y(0)
↦ QIdeal[7]=x(1)^6*y(0)^3-x(1)^6*y(0)^2+x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(1)^2*y(0)
↦ BMap[3]=x(1)^3*y(0)
```

D.5.14.3 jungfib

Procedure from library `resjung.lib` (see Section D.5.14 [resjung_lib], page 1525).

Usage: jungfib(J[,is_noeth]);
J = ideal
j = int

Assume: J = two dimensional ideal

Return: a list l of rings
l[i] is a ring containing two Ideals: QIdeal and BMap. BMap defines a birational morphism from V(QIdeal)–>V(J), such that V(QIdeal) has only quasi-ordinary singularities.
If is_noeth=1 the algorithm assumes J is in noether position with respect to the last two variables. As a default or if is_noeth = 0 the algorithm computes a coordinate change such that J is in noether position. NOTE: since the noether position algorithm is randomized the performance can vary significantly.

Example:

```
LIB "resjung.lib";
//Computing a resolution of singularities of the variety z2-x3-y3
ring r = 0,(x,y,z),dp;
ideal I = z2-x3-y3;
//The ideal is in noether position
list l = jungfib(I,1);
def R1 = l[1];
def R2 = l[2];
setring R1;
QIdeal;
↦ QIdeal[1]=x(2)^6*y(1)^4-x(2)^6*y(1)^3-x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(2)^2*y(1)
↦ BMap[3]=x(2)^3*y(1)^2
setring R2;
QIdeal;
↦ QIdeal[1]=x(1)^6*y(0)^3-x(1)^6*y(0)^2+x^3
BMap;
↦ BMap[1]=x
↦ BMap[2]=x(1)^2*y(0)
↦ BMap[3]=x(1)^3*y(0)
```

D.5.15 resolve_lib

Library: resolve.lib

Purpose: Resolution of singularities (Desingularization) Algorithm of Villamayor

Authors: A. Fruehbis-Krueger, anne@mathematik.uni-kl.de,
G. Pfister, pfister@mathematik.uni-kl.de

References:
[1] J.Kollar: Lectures on Resolution of Singularities, Princeton University Press (2007) (contains large overview over various known methods for curves and surfaces as well as a detailed description of the approach in the general case)
[2] A.Bravo, S.Encinas, O.Villamayor: A Simplified Proof of Desingularisation and Applications, Rev. Math. Iberoamericana 21 (2005), 349-458
(description of the algorithmic proof of desingularization in characteristic zero which underlies this implementation)
[3] A.Fruehbis-Krueger: Computational Aspects of Singularities, in J.-P. Brasselet, J.Damon et al.: Singularities in Geometry and Topology, World Scientific Publishing, 253–327 (2007)
(chapter 4 contains a detailed discussion on algorithmic desingularization and efficiency aspects thereof)

Procedures:

D.5.15.1 blowUp

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: blowUp(J,C[,W][,E]);
W,J,C = ideals,
E = list

Assume: J = ideal containing W (W = 0 if not specified)
C = ideal containing J
E = list of smooth hypersurfaces (e.g. exceptional divisors)

Note: W the ideal of the ambient space, C the ideal of the center of the blowup and J the ideal of the variety
Important difference to blowUp2:
- the ambient space V(W) is blown up and V(J) transformed in it
- V(C) is assumed to be non-singular

Compute: the blowing up of W in C, the exceptional locus, the strict transform of J and the blowup map

Return: list, say l, of size at most size(C),

l[i] is the affine ring corresponding to the i-th chart each l[i] contains the ideals
- aS, ideal of the blownup ambient space
- sT, ideal of the strict transform
- eD, ideal of the exceptional divisor
- bM, ideal corresponding to the blowup map

l[i] also contains a list BO, which can best be viewed with showBO(BO) detailed information on the data type BO can be viewed via the command showDataTypes();

Example:

```
LIB "resolve.lib";
ring R=0,(x,y),dp;
ideal J=x2-y3;
ideal C=x,y;
list blow=blowUp(J,C);
def Q=blow[1];
setring Q;
aS;
↦ aS[1]=0
sT;
↦ sT[1]=y(1)^2-x(2)
eD;
↦ eD[1]=x(2)
bM;
↦ bM[1]=x(2)*y(1)
↦ bM[2]=x(2)
```

D.5.15.2 blowUp2

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: blowUp2(J,C);
J,C = ideals,

Assume: C = ideal containing J

Note: C the ideal of the center of the blowup and J the ideal of the variety
Important differences to blowUp:
- V(J) itself is blown up, not the ambient space
- C is not assumed to be non-singular

Compute: the blowing up of J in C, the exceptional locus and the blow-up map

Return: list, say l, of size at most size(C),

l[i] is the affine ring corresponding to the i-th chart each l[i] contains the ideals
- Jnew, ideal of the blownup J
- eD, ideal of the new exceptional divisor
- bM, ideal corresponding to the blowup map

Example:

```
LIB "resolve.lib";
ring r=0,(x,y,z),dp;
ideal I=z2-x^3*y^2;
ideal C=z,xy;
list li=blowUp2(I,C);
size(li);                    // number of charts
↦ 2
def S1=li[1];
setring S1;                  // chart 1
basering;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(3) y(2)
↦ //        block   2 : ordering C
```

```
Jnew;
↦ Jnew[1]=x(1)*y(2)^2-1
eD;
↦ eD[1]=x(3)
↦ eD[2]=x(1)*y(2)^2-1
bM;
↦ bM[1]=x(1)
↦ bM[2]=x(3)*y(2)^3
↦ bM[3]=x(3)
def S2=li[2];
setring S2;                // chart 2
basering;
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(2) y(1)
↦ //        block   2 : ordering C
Jnew;
↦ Jnew[1]=0
eD;
↦ eD[1]=x(2)*y(1)^2
bM;
↦ bM[1]=y(1)^2
↦ bM[2]=x(2)
↦ bM[3]=x(2)*y(1)^3
```

D.5.15.3 Center

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: Center(J[,W][,E])
J,W = ideals
E = list

Assume: J = ideal containing W (W = 0 if not specified)
E = list of smooth hypersurfaces (e.g. exceptional divisors)

Compute: the center of the blow-up of J for the resolution algorithm of [Bravo,Encinas,Villamayor]

Return: ideal, describing the center

Example:

```
LIB "resolve.lib";
ring R=0,(x,y),dp;
ideal J=x2-y3;
Center(J);
↦ _[1]=y
↦ _[2]=x
```

D.5.15.4 resolve

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: resolve (J); or resolve (J,i[,k]);
J ideal
i,k int

Compute: a resolution of J,
if i > 0 debugging is turned on according to the following switches:
j1: value 0 or 1; turn off or on correctness checks in all steps
j2: value 0 or 2; turn off or on debugCenter
j3: value 0 or 4; turn off or on debugBlowUp
j4: value 0 or 8; turn off or on debugCoeff
j5: value 0 or 16:turn off or on debugging of Intersection with E^-
j6: value 0 or 32:turn off or on stop after pass through the loop
i=j1+j2+j3+j4+j5+j6

Return: a list l of 2 lists of rings
l[1][i] is a ring containing a basic object BO, the result of the resolution.
l[2] contains all rings which occurred during the resolution process

Note: result may be viewed in a human readable form using presentTree()

Example:

```
LIB "resolve.lib";
ring R=0,(x,y,z),dp;
ideal J=x3+y5+yz2+xy4;
list L=resolve(J,0);
def Q=L[1][7];
setring Q;
showBO(BO);
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(3)^2*y(1)+x(1)^2+y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)
↦ [5]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^6*x(3)^5*y(1)^2
↦ _[2]=x(1)^4*x(3)^3*y(1)
↦ _[3]=x(1)^7*x(3)^6*y(1)^2
↦
```

D.5.15.5 showBO

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: showBO(BO);
BO=basic object, a list: ideal W,

ideal J,
intvec b (already truncated for Coeff),
list Ex (already truncated for Coeff),
ideal ab,
intvec v,
intvec w (already truncated for Coeff),
matrix M

Return: nothing, only pretty printing

D.5.15.6 presentTree

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: presentTree(L);
L=list, output of resolve

Return: nothing, only pretty printing of the output data of resolve()

D.5.15.7 showDataTypes

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: showDataTypes();

Return: nothing, only pretty printing of extended version of help text

D.5.15.8 blowUpBO

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: blowUpBO (BO,C,e);
BO = basic object, a list: ideal W,
ideal J,
intvec b,
list Ex,
ideal ab,
intvec v,
intvec w,
matrix M
C = ideal
e = integer (0 usual blowing up, 1 deleting extra charts, 2 deleting no charts)

Assume: R = basering, a polynomial ring, W an ideal of R,
J = ideal containing W,
C = ideal containing J

Compute: the blowing up of BO[1] in C, the exceptional locus, the strict transform of BO[2]

Note: blowUpBO may be applied to basic objects in the sense of [Bravo, Encinas, Villamayor] in the following referred to as BO and to presentations in the sense of [Bierstone, Milman] in the following referred to as BM.

Return: a list l of length at most size(C),

l[i] is a ring containing an object BO resp. BM:

BO[1]=BM[1] an ideal, say Wi, defining the ambient space of the i-th chart of the blowing up

BO[2]=BM[2] an ideal defining the strict transform

BO[3] intvec, the first integer b such that in the original object (Delta^b(BO[2]))==1 the subsequent integers have the same property for Coeff-Objects of BO[2] used when determining the center

BM[3] intvec, BM[3][i] is the assigned multiplicity of BM[2][i]

BO[4]=BM[4] the list of exceptional divisors

BO[5]=BM[5] an ideal defining the map K[W] —-> K[Wi]

BO[6]=BM[6] an intvec BO[6][j]=1 indicates that <BO[4][j],BO[2]>=1, i.e. the strict transform does not meet the j-th exceptional divisor

BO[7] intvec, the index of the first blown-up object in the resolution process leading to this object for which the value of b was BO[3] the subsequent ones are the indices for the Coeff-Objects of BO[2] used when determining the center

BM[7] intvec, BM[7][i] is the index at which the (2i-1)st entry of the invariant first reached its current maximal value

BO[8]=BM[8] a matrix indicating that BO[4][i] meets BO[4][j] by BO[8][i,j]=1 for i < j

BO[9] empty

BM[9] the invariant

Example:

```
LIB "resolve.lib";
ring R=0,(x,y),dp;
ideal W;
ideal J=x2-y3;
intvec b=1;
list E;
ideal abb=maxideal(1);
intvec v;
intvec w=-1;
matrix M;
intvec ma;
list BO=W,J,b,E,abb,v,w,M,ma;
ideal C=CenterBO(BO)[1];
list blow=blowUpBO(BO,C,0);
def Q=blow[1];
setring Q;
BO;
↦ [1]:
↦    _[1]=0
↦ [2]:
↦    _[1]=y(1)^2-x(2)
↦ [3]:
↦    1
↦ [4]:
↦    [1]:
↦       _[1]=x(2)
↦ [5]:
↦    _[1]=x(2)*y(1)
↦    _[2]=x(2)
↦ [6]:
↦    0
↦ [7]:
↦    -1
↦ [8]:
↦    _[1,1]=0
↦ [9]:
↦    0
```

D.5.15.9 createBO

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: createBO(J[,W][,E]);
J,W = ideals
E = list

Assume: J = ideal containing W (W = 0 if not specified)
E = list of smooth hypersurfaces (e.g. exceptional divisors)

Return: list BO representing a basic object :
BO[1] ideal W, if W has been specified; ideal(0) otherwise BO[2] ideal J
BO[3] intvec
BO[4] the list E of exceptional divisors if specified; empty list otherwise
BO[5] an ideal defining the identity map
BO[6] an intvec
BO[7] intvec
BO[8] a matrix
entries 3,5,6,7,8 are initialized appropriately for use of CenterBO and blowUpBO

Example:

```
LIB "resolve.lib";
ring R=0,(x,y,z),dp;
ideal J=x2-y3;
createBO(J,ideal(z));
↦ [1]:
↦    _[1]=z
↦ [2]:
↦    _[1]=-y3+x2
↦ [3]:
↦    0
↦ [4]:
↦    empty list
↦ [5]:
↦    _[1]=x
↦    _[2]=y
↦    _[3]=z
↦ [6]:
↦    0
↦ [7]:
↦    -1
↦ [8]:
↦    _[1,1]=0
```

D.5.15.10 CenterBO

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: CenterBO(BO);
BO = basic object, a list: ideal W,
ideal J,
intvec b,
list Ex,
ideal ab,
intvec v,
intvec w,
matrix M

Assume: R = basering, a polynomial ring, W an ideal of R,
J = ideal containing W

Compute: the center of the next blow-up of BO in the resolution algorithm of [Bravo,Encinas,Villamayor]

Return: list l,
l[1]: ideal describing the center
l[2]: intvec w obtained in the process of determining l[1]
l[3]: intvec b obtained in the process of determining l[1]
l[4]: intvec inv obtained in the process of determining l[1]

Example:

```
LIB "resolve.lib";
ring R=0,(x,y),dp;
ideal W;
ideal J=x2-y3;
intvec b=1;
list E;
ideal abb=maxideal(1);
intvec v;
intvec w=-1;
matrix M;
list BO=W,J,b,E,abb,v,w,M,v;
CenterBO(BO);
↦ [1]:
↦    _[1]=y
↦    _[2]=x
↦ [2]:
↦    -1
↦ [3]:
↦    2
↦ [4]:
↦    0
```

D.5.15.11 Delta

Procedure from library `resolve.lib` (see Section D.5.15 [resolve_lib], page 1527).

Usage: Delta (BO);
BO = basic object, a list: ideal W,

ideal J,
intvec b,
list Ex,
ideal ab,
intvec v,
intvec w,
matrix M

Assume: R = basering, a polynomial ring, W an ideal of R,
J = ideal containing W

Compute: Delta-operator applied to J in the notation of
[Bravo,Encinas,Villamayor]

Return: ideal

Example:

```
LIB "resolve.lib";
ring R=0,(x,y,z),dp;
ideal W=z^2-x;
ideal J=x*y^2+x^3;
intvec b=1;
list E;
ideal abb=maxideal(1);
intvec v;
intvec w=-1;
matrix M;
list BO=W,J,b,E,abb,v,w,M;
Delta(BO);
↦ _[1]=z2-x
↦ _[2]=xy
↦ _[3]=3x2z+y2z
↦ _[4]=x3
```

D.5.15.12 DeltaList

Procedure from library resolve.lib (see Section D.5.15 [resolve_lib], page 1527).

Usage: DeltaList (BO);
BO = basic object, a list: ideal W,
ideal J,
intvec b,
list Ex,
ideal ab,
intvec v,
intvec w,
matrix M

Assume: R = basering, a polynomial ring, W an ideal of R,
J = ideal containing W

Compute: Delta-operator iteratively applied to J in the notation of [Bravo,Encinas,Villamayor]

Return: list l of length ((max w-ord) * b),
l[i+1]=Delta^i(J)

Example:

```
LIB "resolve.lib";
ring R=0,(x,y,z),dp;
ideal W=z^2-x;
ideal J=x*y^2+x^3;
intvec b=1;
list E;
ideal abb=maxideal(1);
intvec v;
intvec w=-1;
matrix M;
list BO=W,J,b,E,abb,v,w,M;
DeltaList(BO);
↦ [1]:
↦    _[1]=x3+xy2
↦ [2]:
↦    _[1]=z2-x
↦    _[2]=xy
↦    _[3]=3x2z+y2z
↦    _[4]=x3
↦ [3]:
↦    _[1]=x
↦    _[2]=z2
↦    _[3]=yz
↦    _[4]=y2
↦ [4]:
↦    _[1]=z
↦    _[2]=y
↦    _[3]=x
```

D.5.16 reszeta_lib

Library: reszeta.lib

Purpose: topological Zeta-function and some other applications of desingularization

Authors: A. Fruehbis-Krueger, anne@mathematik.uni-kl.de,
G. Pfister, pfister@mathematik.uni-kl.de

References:

[1] Fruehbis-Krueger,A., Pfister,G.: Some Applications of Resolution of Singularities from a Practical Point of View, in Computational Commutative and Non-commutative Algebraic Geometry,
NATO Science Series III, Computer and Systems Sciences 196, 104-117 (2005) [2] Fruehbis-Krueger: An Application of Resolution of Singularities:
Computing the topological Zeta-function of isolated surface singularities in (C^3,0), in D.Cheniot, N.Dutertre et al.(Editors): Singularity Theory,
World Scientific Publishing (2007)

Procedures:

D.5.16.1 intersectionDiv

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: intersectionDiv(L);
L = list of rings

Assume: L is output of resolution of singularities
(only case of isolated surface singularities)

Compute: intersection matrix and genera of the exceptional divisors (considered as curves on the strict transform)

Return: list l, where
l[1]: intersection matrix of exceptional divisors
l[2]: intvec, genera of exceptional divisors
l[3]: divisorList, encoding the identification of the divisors

Example:

```
LIB "reszeta.lib";
ring r = 0,(x(1..3)),dp(3);
ideal J=x(3)^5+x(2)^4+x(1)^3+x(1)*x(2)*x(3);
list re=resolve(J);
list di=intersectionDiv(re);
di;
↦ [1]:
↦    -3,1,1,
↦    1,-4,1,
↦    1,1,-2
↦ [2]:
↦    0,0,0
↦ [3]:
↦    [1]:
↦       [1]:
↦          3,1,1
↦       [2]:
↦          5,1,1
↦    [2]:
↦       [1]:
↦          3,1,2
↦       [2]:
↦          4,1,1
↦       [3]:
↦          5,1,2
↦    [3]:
↦       [1]:
↦          4,2,1
↦       [2]:
↦          5,2,1
↦ [4]:
↦    1,1,1
```

D.5.16.2 spectralNeg

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: spectralNeg(L);
L = list of rings

Assume: L is output of resolution of singularities

Return: list of numbers, each a spectral number in (-1,0]

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=x3+y4+z5;
list L=resolve(I,"K");
spectralNeg(L);
↦ [1]:
↦ -13/60
↦ [2]:
↦ -1/60
LIB"gmssing.lib";
ring r=0,(x,y,z),ds;
poly f=x3+y4+z5;
spectrum(f);
↦ [1]:
↦    _[1]=-13/60
↦    _[2]=-1/60
↦    _[3]=1/30
↦    _[4]=7/60
↦    _[5]=11/60
↦    _[6]=7/30
↦    _[7]=17/60
↦    _[8]=19/60
↦    _[9]=11/30
↦    _[10]=23/60
↦    _[11]=13/30
↦    _[12]=29/60
↦    _[13]=31/60
↦    _[14]=17/30
↦    _[15]=37/60
↦    _[16]=19/30
↦    _[17]=41/60
↦    _[18]=43/60
↦    _[19]=23/30
↦    _[20]=49/60
↦    _[21]=53/60
↦    _[22]=29/30
↦    _[23]=61/60
↦    _[24]=73/60
↦ [2]:
↦    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

D.5.16.3 discrepancy

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: discrepancy(L);
L = list of rings

Assume: L is the output of resolution of singularities

Return: discrepancies of the given resolution

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=x2+y2+z3;
list re=resolve(I);
discrepancy(re);
↦ 0,1,1
```

D.5.16.4 zetaDL

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Assume: L is the output of resolution of singularities

Compute: local Denef-Loeser zeta function, if string s1 is present and has the value 'local'; global Denef-Loeser zeta function otherwise
if string s1 or s2 has the value `"A"`, additionally the characteristic polynomial of the monodromy is computed

Return: list l
if a is not present:
l[1]: string specifying the top. zeta function
l[2]: string specifying characteristic polynomial of monodromy, if `"A"` was specified
if a is present:
l[1]: string specifying the top. zeta function
l[2]: list ast,
ast[1]=chi(Ei^*)
ast[2]=chi(Eij^*)
ast[3]=chi(Eijk^*)
l[3]: intvec nu of multiplicities as needed in computation of zeta function
l[4]: intvec N of multiplicities as needed in computation of zeta function
l[5]: string specifying characteristic polynomial of monodromy, if `"A"` was specified

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=x2+y2+z3;
list re=resolve(I,"K");
zetaDL(re,1);
↦ [1]:
↦    (s+4)/(3s2+7s+4)
I=(xz+y2)*(xz+y2+x2)+z5;
list L=resolve(I,"K");
zetaDL(L,1);
↦ [1]:
↦    (20s2+130s+87)/(160s3+396s2+323s+87)
//===== expected zeta function =========
// (20s^2+130s+87)/((1+s)*(3+4s)*(29+40s))
//======================================
```

D.5.16.5 collectDiv

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: collectDiv(L);
L = list of rings

Assume: L is output of resolution of singularities

Compute: list representing the identification of the exceptional divisors in the various charts

Return: list l, where
l[1]: intmat, entry k in position i,j implies BO[4][j] of chart i is divisor k (if k!=0)
if k==0, no divisor corresponding to i,j
l[2]: list ll, where each entry of ll is a list of intvecs entry i,j in list ll[k] implies BO[4][j] of chart i
is divisor k
l[3]: list L

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=xyz+x4+y4+z4;
//we really need to blow up curves even if the generic point of
//the curve the total transform is n.c.
//this occurs here in r[2][5]
list re=resolve(I);
list di=collectDiv(re);
di[1];
↦ 0,0,0,
↦ 1,0,0,
↦ 1,0,0,
↦ 1,0,0,
↦ 1,2,0,
↦ 1,2,0,
↦ 1,3,0,
↦ 1,3,0,
↦ 1,4,0,
↦ 1,4,0,
↦ 0,2,5,
↦ 1,0,5,
↦ 0,2,5,
↦ 1,0,5,
↦ 0,3,6,
↦ 1,0,6,
↦ 0,3,6,
↦ 1,0,6,
↦ 0,4,7,
↦ 1,0,7,
↦ 0,4,7,
↦ 1,0,7
di[2];
↦ [1]:
↦    [1]:
↦       2,1
↦    [2]:
↦       3,1
↦    [3]:
↦       4,1
↦    [4]:
↦       5,1
↦    [5]:
↦       6,1
↦    [6]:
↦       7,1
↦    [7]:
↦       8,1
↦    [8]:
↦       9,1
↦    [9]:
↦       10,1
↦    [10]:
↦       12,1
↦    [11]:
↦       14,1
↦    [12]:
↦       16,1
↦    [13]:
↦       18,1
↦    [14]:
↦       20,1
↦    [15]:
↦       22,1
↦ [2]:
↦    [1]:
↦       5,2
↦    [2]:
↦       6,2
↦    [3]:
↦       11,2
↦    [4]:
↦       13,2
↦ [3]:
↦    [1]:
↦       7,2
↦    [2]:
↦       8,2
↦    [3]:
↦       15,2
↦    [4]:
↦       17,2
↦ [4]:
↦    [1]:
↦       9,2
↦    [2]:
↦       10,2
↦    [3]:
↦       19,2
↦    [4]:
↦       21,2
↦ [5]:
↦    [1]:
↦       11,3
↦    [2]:
```

```
↦       12,3
↦    [3]:
↦       13,3
↦    [4]:
↦       14,3
↦ [6]:
↦    [1]:
↦       15,3
↦    [2]:
↦       16,3
↦    [3]:
↦       17,3
↦    [4]:
↦       18,3
↦ [7]:
↦    [1]:
↦       19,3
↦    [2]:
↦       20,3
↦    [3]:
↦       21,3
↦    [4]:
↦       22,3
↦ [8]:
↦    [1]:
↦       11,0
↦    [2]:
↦       12,0
↦    [3]:
↦       13,0
↦    [4]:
↦       14,0
↦    [5]:
↦       15,0
↦    [6]:
↦       16,0
↦    [7]:
↦       17,0
↦    [8]:
↦       18,0
↦    [9]:
↦       19,0
↦    [10]:
↦       20,0
↦    [11]:
↦       21,0
↦    [12]:
↦       22,0
```

D.5.16.6 prepEmbDiv

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: prepEmbDiv(L[,a]);
L = list of rings
a = integer

Assume: L is output of resolution of singularities

Compute: if a is not present: exceptional divisors including components of the strict transform
otherwise: only exceptional divisors

Return: list of Q-irreducible exceptional divisors (embedded case)

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=x2+y2+z11;
list L=resolve(I);
prepEmbDiv(L);
↦ [1]:
↦    [1]:
↦       2,1
↦    [2]:
↦       3,1
↦    [3]:
↦       4,1
↦ [2]:
↦    [1]:
↦       4,2
↦    [2]:
↦       5,2
↦    [3]:
↦       6,2
↦ [3]:
↦    [1]:
↦       6,3
↦    [2]:
↦       7,3
↦    [3]:
↦       8,3
↦ [4]:
↦    [1]:
↦       8,4
↦    [2]:
↦       9,4
↦    [3]:
↦       10,4
↦ [5]:
↦    [1]:
↦       10,5
↦    [2]:
↦       11,5
↦    [3]:
↦       12,5
↦    [4]:
↦       13,5
↦    [5]:
```

```
↦        15,5
↦     [6]:
↦        17,5
↦  [6]:
↦     [1]:
↦        12,6
↦     [2]:
↦        13,6
↦     [3]:
↦        14,6
↦     [4]:
↦        16,6
↦  [7]:
↦     [1]:
↦        14,7
↦     [2]:
↦        15,7
↦     [3]:
↦        16,7
↦     [4]:
↦        17,7
↦  [8]:
↦     [1]:
↦        3,2
↦     [2]:
↦        4,3
↦     [3]:
↦        6,4
↦     [4]:
↦        8,5
↦     [5]:
↦        10,6
↦     [6]:
↦        14,8
↦     [7]:
↦        15,8
↦     [8]:
↦        16,8
↦     [9]:
↦        17,8
```

D.5.16.7 abstractR

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: abstractR(L);
L = list of rings

Assume: L is output of resolution of singularities

Note: currently only implemented for isolated surface singularities

Return: list l
l[1]: intvec, where
l[1][i]=1 if the corresponding ring is a final chart

of non-embedded resolution
l[1][i]=0 otherwise
l[2]: intvec, where
l[2][i]=1 if the corresponding ring does not occur
in the non-embedded resolution
l[2][i]=0 otherwise
l[3]: list L

Example:

```
LIB "reszeta.lib";
ring r = 0,(x,y,z),dp;
ideal I=x2+y2+z11;
list L=resolve(I);
list absR=abstractR(L);
absR[1];
↦ 0,0,1,1,0,1,0,1,0,1,1,0,0,0,0,0,0
absR[2];
↦ 0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1
```

D.5.16.8 computeV

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: computeV(L,DL);
L = list of rings
DL = divisor list

Assume: L has structure of output of resolve
DL has structure of output of prepEmbDiv

Return: intvec,
i-th entry is multiplicity of i-th divisor in
pullback of volume form

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=(x-y)*(x-z)*(y-z)-z4;
list re=resolve(I,1);
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 1
↦ Index of the current chart in chart-tree: 1
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=-z4+x2y-xy2-x2z+y2z+xz2-yz2
↦
↦ ==== Exceptional Divisors:
↦ empty list
↦
```

```
↦ ==== Images of variables of original ring:
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=z
↦ _[2]=y
↦ _[3]=x
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 4
↦ Index of the current chart in chart-tree: 2
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=y(1)^2*y(2)-y(1)*y(2)^2-y(1)^2+y(2)^2+x(3)+y(1)-y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)*y(2)
↦ _[2]=x(3)*y(1)
↦ _[3]=x(3)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)-1
↦ _[2]=y(1)-1
↦ _[3]=x(3)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 6
↦ Index of the current chart in chart-tree: 3
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)*y(0)^4-y(0)^2*y(2)+y(0)*y(2)^2+y(0)^2-y(2)^2-y(0)+y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(2)
↦
↦ ==== Images of variables of original ring:
```

```
↦ _[1]=x(2)*y(2)
↦ _[2]=x(2)
↦ _[3]=x(2)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)-1
↦ _[2]=y(0)-1
↦ _[3]=x(2)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 8
↦ Index of the current chart in chart-tree: 4
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*y(0)^4+y(0)^2*y(1)-y(0)*y(1)^2-y(0)^2+y(1)^2+y(0)-y(1)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(1)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)
↦ _[2]=x(1)*y(1)
↦ _[3]=x(1)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(1)-1
↦ _[2]=y(0)-1
↦ _[3]=x(1)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 10
↦ Index of the current chart in chart-tree: 5
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^2*y(1)^2-x(3)^2*y(1)-2*x(3)*y(1)^2+2*x(3)*y(1)+y(1)^2-y(1)+y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(3)-1
↦
```

```
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(2)-x(3)*y(2)
↦ _[2]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
↦ _[3]=x(3)*y(2)-y(2)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 12
↦ Index of the current chart in chart-tree: 6
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(0)^2-x(2)^2*y(0)-2*x(2)*y(0)^2+2*x(2)*y(0)+y(0)^2-y(0)-y(2)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[2]=x(2)^2*y(2)-x(2)*y(2)
↦ _[3]=x(2)*y(2)-y(2)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 14
↦ Index of the current chart in chart-tree: 7
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4*y(2)-4*x(3)^3*y(1)^4*y(2)+4*x(3)^3*y(1)^3*y(2)+6*x(3)^\
  2*y(1)^4*y(2)-12*x(3)^2*y(1)^3*y(2)-4*x(3)*y(1)^4*y(2)+6*x(3)^2*y(1)^2*y(\
  2)+12*x(3)*y(1)^3*y(2)+y(1)^4*y(2)-x(3)^2*y(1)^2-12*x(3)*y(1)^2*y(2)-4*y(\
  1)^3*y(2)+x(3)^2*y(1)+2*x(3)*y(1)^2+4*x(3)*y(1)*y(2)+6*y(1)^2*y(2)-2*x(3)\
  *y(1)-y(1)^2-4*y(1)*y(2)+y(1)+y(2)
↦ 
↦ ==== Exceptional Divisors:
```

```
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(3)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(2)-x(3)*y(2)
↦ _[2]=x(3)*y(2)-y(2)
↦ _[3]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 16
↦ Index of the current chart in chart-tree: 8
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*y(2)+x(2)^2*y(0)^2-x(2)^2*y(0)-2*x(2)*y(0)^2+2*x(2)*y(0)+y(0)\
   ^2-y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[2]=x(2)*y(2)-y(2)
↦ _[3]=x(2)^2*y(2)-x(2)*y(2)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 18
↦ Index of the current chart in chart-tree: 9
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4*y(2)-4*x(3)^3*y(1)^4*y(2)+4*x(3)^3*y(1)^3*y(2)+6*x(3)^\
   2*y(1)^4*y(2)-12*x(3)^2*y(1)^3*y(2)-4*x(3)*y(1)^4*y(2)+6*x(3)^2*y(1)^2*y(\
   2)+12*x(3)*y(1)^3*y(2)+y(1)^4*y(2)+x(3)^2*y(1)^2-12*x(3)*y(1)^2*y(2)-4*y(\
   1)^3*y(2)-x(3)^2*y(1)-2*x(3)*y(1)^2+4*x(3)*y(1)*y(2)+6*y(1)^2*y(2)+2*x(3)\
   *y(1)+y(1)^2-4*y(1)*y(2)-y(1)+y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(3)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)*y(2)-y(2)
↦ _[2]=x(3)^2*y(2)-x(3)*y(2)
↦ _[3]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 20
↦ Index of the current chart in chart-tree: 10
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*y(2)-x(2)^2*y(0)^2+x(2)^2*y(0)+2*x(2)*y(0)^2-2*x(2)*y(0)-y(0)\
   ^2+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*y(2)-y(2)
↦ _[2]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[3]=x(2)^2*y(2)-x(2)*y(2)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 22
↦ Index of the current chart in chart-tree: 11
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^2*y(1)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 22
↦ Index of the current chart in chart-tree: 12
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[2]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦ _[3]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 24
↦ Index of the current chart in chart-tree: 13
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2-1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^2*y(1)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 24
↦ Index of the current chart in chart-tree: 14
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)-y(0)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦ _[2]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[3]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
```

```
↦
↦ ------------------------- Upcoming Center ----------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 26
↦ Index of the current chart in chart-tree: 15
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
  (2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^2*y(1)
↦ _[3]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦
↦ ------------------------- Upcoming Center ----------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
  (2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 26
↦ Index of the current chart in chart-tree: 16
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)-x(1)\
  *x(2)^2+4*x(1)*x(2)*y(0)+6*x(2)^2*y(0)+x(1)*x(2)+x(2)^2-4*x(2)*y(0)-x(2)+\
  y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
```

```
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[2]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[3]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 28
↦ Index of the current chart in chart-tree: 17
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+6*\
   x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^2*y(1)
↦ _[3]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+6*\
   x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 28
↦ Index of the current chart in chart-tree: 18
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
```

```
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)+x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦ _[2]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[3]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 30
↦ Index of the current chart in chart-tree: 19
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
   (2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(1)
↦ _[2]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
   (2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 30
↦ Index of the current chart in chart-tree: 20
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
   2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
   0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+x(1)\
   *x(2)^2+4*x(1)*x(2)*y(0)+6*x(2)^2*y(0)-x(1)*x(2)-x(2)^2-4*x(2)*y(0)+x(2)+\
   y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[2]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[3]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ --------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 32
↦ Index of the current chart in chart-tree: 21
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3-x(2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+6*\
   x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
```

```
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(1)
↦ _[2]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3-x(2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+6*\
  x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 32
↦ Index of the current chart in chart-tree: 22
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)-x(1)*x(2)^2+x(1)*x(2)+x(2)^2-x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[2]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦ _[3]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 23
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)+1
↦
```

```
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^3*y(1)^2
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)+1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 1
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 24
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2-x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[2]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[3]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2-x(2)+y(0)
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 2
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 25
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)-1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 3
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 26
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2-x(2)-y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
   *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[2]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[3]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
```

```
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2-x(2)-y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 4
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 27
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
  (2)^2*y(1)+4*x(2)*x(3)*y(1)+x(2)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^3*y(1)^2
↦ _[3]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
  (2)^2*y(1)+4*x(2)*x(3)*y(1)+x(2)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 5
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 28
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
  1)*x(2)*y(0)+6*x(2)^2*y(0)-x(2)^2-4*x(2)*y(0)+x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
```

```
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[2]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[3]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
  1)*x(2)*y(0)+6*x(2)^2*y(0)-x(2)^2-4*x(2)*y(0)+x(2)+y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 6
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 29
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2+x(2)^2*y(1)-x(2)*y(1)+\
  4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^3*y(1)^2
↦ _[3]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2+x(2)^2*y(1)-x(2)*y(1)+\
  4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 7
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 30
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)+x(2)^2-x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
   *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[2]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[3]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*y(0)+x(2)^2-x(2)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 8
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 31
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
   (2)^2*y(1)+4*x(2)*x(3)*y(1)-x(2)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
```

```
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2
↦ _[2]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
  (2)^2*y(1)+4*x(2)*x(3)*y(1)-x(2)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 9
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 32
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
  1)*x(2)*y(0)+6*x(2)^2*y(0)+x(2)^2-4*x(2)*y(0)-x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[2]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[3]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
  1)*x(2)*y(0)+6*x(2)^2*y(0)+x(2)^2-4*x(2)*y(0)-x(2)+y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 10
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 33
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
```

```
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2-x(2)^2*y(1)+x(2)*y(1)+\
   4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2
↦ _[2]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2-x(2)^2*y(1)+x(2)*y(1)+\
   4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 11
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 34
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)-x(2)^2+x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[2]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
   *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[3]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦
```

```
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*y(0)-x(2)^2+x(2)
↦ ----------------------------------------------------------------
↦ ============= result will be tested ==========
↦
↦ the number of charts obtained: 12
↦ =============     result is o.k.    ==========
list iden=prepEmbDiv(re);
intvec v=computeV(re, iden);
v;
↦ 3,5,8,11,1
```

D.5.16.9 computeN

Procedure from library `reszeta.lib` (see Section D.5.16 [reszeta_lib], page 1537).

Usage: computeN(L,DL);
L = list of rings
DL = divisor list

Assume: L has structure of output of resolve
DL has structure of output of prepEmbDiv

Return: intvec, i-th entry is multiplicity of i-th divisor
in total transform under resolution

Example:

```
LIB "reszeta.lib";
ring R=0,(x,y,z),dp;
ideal I=(x-y)*(x-z)*(y-z)-z4;
list re=resolve(I,1);
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 1
↦ Index of the current chart in chart-tree: 1
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=-z4+x2y-xy2-x2z+y2z+xz2-yz2
↦
↦ ==== Exceptional Divisors:
↦ empty list
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=z
↦ _[2]=y
↦ _[3]=x
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 4
↦ Index of the current chart in chart-tree: 2
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=y(1)^2*y(2)-y(1)*y(2)^2-y(1)^2+y(2)^2+x(3)+y(1)-y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)*y(2)
↦ _[2]=x(3)*y(1)
↦ _[3]=x(3)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)-1
↦ _[2]=y(1)-1
↦ _[3]=x(3)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 6
↦ Index of the current chart in chart-tree: 3
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)*y(0)^4-y(0)^2*y(2)+y(0)*y(2)^2+y(0)^2-y(2)^2-y(0)+y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(2)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*y(2)
↦ _[2]=x(2)
↦ _[3]=x(2)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)-1
↦ _[2]=y(0)-1
↦ _[3]=x(2)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 8
↦ Index of the current chart in chart-tree: 4
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*y(0)^4+y(0)^2*y(1)-y(0)*y(1)^2-y(0)^2+y(1)^2+y(0)-y(1)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=x(1)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)
↦ _[2]=x(1)*y(1)
↦ _[3]=x(1)*y(0)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(1)-1
↦ _[2]=y(0)-1
↦ _[3]=x(1)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 10
↦ Index of the current chart in chart-tree: 5
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^2*y(1)^2-x(3)^2*y(1)-2*x(3)*y(1)^2+2*x(3)*y(1)+y(1)^2-y(1)+y(2)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(3)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(2)-x(3)*y(2)
↦ _[2]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
↦ _[3]=x(3)*y(2)-y(2)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 12
↦ Index of the current chart in chart-tree: 6
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(0)^2-x(2)^2*y(0)-2*x(2)*y(0)^2+2*x(2)*y(0)+y(0)^2-y(0)-y(2)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[2]=x(2)^2*y(2)-x(2)*y(2)
↦ _[3]=x(2)*y(2)-y(2)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 14
↦ Index of the current chart in chart-tree: 7
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4*y(2)-4*x(3)^3*y(1)^4*y(2)+4*x(3)^3*y(1)^3*y(2)+6*x(3)^\
   2*y(1)^4*y(2)-12*x(3)^2*y(1)^3*y(2)-4*x(3)*y(1)^4*y(2)+6*x(3)^2*y(1)^2*y(\
   2)+12*x(3)*y(1)^3*y(2)+y(1)^4*y(2)-x(3)^2*y(1)^2-12*x(3)*y(1)^2*y(2)-4*y(\
   1)^3*y(2)+x(3)^2*y(1)+2*x(3)*y(1)^2+4*x(3)*y(1)*y(2)+6*y(1)^2*y(2)-2*x(3)\
   *y(1)-y(1)^2-4*y(1)*y(2)+y(1)+y(2)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(3)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(2)-x(3)*y(2)
↦ _[2]=x(3)*y(2)-y(2)
↦ _[3]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
```

```
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 16
↦ Index of the current chart in chart-tree: 8
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*y(2)+x(2)^2*y(0)^2-x(2)^2*y(0)-2*x(2)*y(0)^2+2*x(2)*y(0)+y(0)\
  ^2-y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[2]=x(2)*y(2)-y(2)
↦ _[3]=x(2)^2*y(2)-x(2)*y(2)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 18
↦ Index of the current chart in chart-tree: 9
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4*y(2)-4*x(3)^3*y(1)^4*y(2)+4*x(3)^3*y(1)^3*y(2)+6*x(3)^\
  2*y(1)^4*y(2)-12*x(3)^2*y(1)^3*y(2)-4*x(3)*y(1)^4*y(2)+6*x(3)^2*y(1)^2*y(\
  2)+12*x(3)*y(1)^3*y(2)+y(1)^4*y(2)+x(3)^2*y(1)^2-12*x(3)*y(1)^2*y(2)-4*y(\
  1)^3*y(2)-x(3)^2*y(1)-2*x(3)*y(1)^2+4*x(3)*y(1)*y(2)+6*y(1)^2*y(2)+2*x(3)\
  *y(1)+y(1)^2-4*y(1)*y(2)-y(1)+y(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
```

```
↦    _[1]=x(3)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)*y(2)-y(2)
↦ _[2]=x(3)^2*y(2)-x(3)*y(2)
↦ _[3]=x(3)^2*y(1)*y(2)-2*x(3)*y(1)*y(2)+x(3)*y(2)+y(1)*y(2)-y(2)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(3)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 20
↦ Index of the current chart in chart-tree: 10
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*y(2)-x(2)^2*y(0)^2+x(2)^2*y(0)+2*x(2)*y(0)^2-2*x(2)*y(0)-y(0)\
   ^2+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(2)
↦ [2]:
↦    _[1]=x(2)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*y(2)-y(2)
↦ _[2]=x(2)^2*y(0)*y(2)-2*x(2)*y(0)*y(2)+x(2)*y(2)+y(0)*y(2)-y(2)
↦ _[3]=x(2)^2*y(2)-x(2)*y(2)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(2)
↦ _[2]=x(2)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 22
↦ Index of the current chart in chart-tree: 11
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
```

```
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^2*y(1)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 22
↦ Index of the current chart in chart-tree: 12
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)+y(0)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[2]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦ _[3]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 24
↦ Index of the current chart in chart-tree: 13
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
```

```
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2-1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^2*y(1)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 24
↦ Index of the current chart in chart-tree: 14
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)-y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦ _[2]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[3]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 26
↦ Index of the current chart in chart-tree: 15
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
   (2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^2*y(1)
↦ _[3]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
   (2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 26
↦ Index of the current chart in chart-tree: 16
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
   2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
   0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)-x(1)\
   *x(2)^2+4*x(1)*x(2)*y(0)+6*x(2)^2*y(0)+x(1)*x(2)+x(2)^2-4*x(2)*y(0)-x(2)+\
   y(0)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[2]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[3]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 28
↦ Index of the current chart in chart-tree: 17
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+6*\
  x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[2]=x(3)^2*y(1)
↦ _[3]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+x(2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+6*\
  x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 28
↦ Index of the current chart in chart-tree: 18
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)+x(1)*x(2)^2-x(1)*x(2)-x(2)^2+x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
```

```
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
   (1)*y(0)-x(2)*y(0)+y(0)
↦ _[2]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[3]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 30
↦ Index of the current chart in chart-tree: 19
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
   (2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(1)
↦ _[2]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
   (2)^2*x(3)*y(1)^2-x(2)*x(3)*y(1)^2+4*x(2)*x(3)*y(1)+1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 30
↦ Index of the current chart in chart-tree: 20
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
```

```
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+x(1)\
  *x(2)^2+4*x(1)*x(2)*y(0)+6*x(2)^2*y(0)-x(1)*x(2)-x(2)^2-4*x(2)*y(0)+x(2)+\
  y(0)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[2]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ _[3]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 32
↦ Index of the current chart in chart-tree: 21
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3-x(2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+6*\
  x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=y(1)
↦ [3]:
↦    _[1]=x(3)
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^2*y(1)
↦ _[2]=x(2)*x(3)^3*y(1)^2+x(3)^2*y(1)
↦ _[3]=x(3)^3*y(1)^2+x(3)^2*y(1)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3-x(2)^2*x(3)*y(1)^2+x(2)*x(3)*y(1)^2+6*\
  x(3)^2*y(1)^2+4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 32
↦ Index of the current chart in chart-tree: 22
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)-x(1)*x(2)^2+x(1)*x(2)+x(2)^2-x(2)
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=x(1)-1
↦ 
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^2*y(0)-2*x(1)*y(0)+y(0)
↦ _[2]=x(1)^3*x(2)*y(0)-3*x(1)^2*x(2)*y(0)+x(1)^2*y(0)+3*x(1)*x(2)*y(0)-2*x\
  (1)*y(0)-x(2)*y(0)+y(0)
↦ _[3]=x(1)^3*y(0)-2*x(1)^2*y(0)+x(1)*y(0)
↦ 
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=y(0)
↦ _[2]=x(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 0
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 23
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦ 
↦ ==== Ambient Space:
↦ _[1]=0
↦ 
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)+1
↦ 
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
```

```
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 1
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 24
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2-x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[2]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[3]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2-x(2)+y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 2
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 25
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)-1
↦
↦ ==== Exceptional Divisors:
```

```
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2*y(1)-x(2)*y(1)-1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 3
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 26
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^2-x(2)-y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[2]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[3]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^2-x(2)-y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 4
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 27
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
   (2)^2*y(1)+4*x(2)*x(3)*y(1)+x(2)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^3*y(1)^2
↦ _[3]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2-x\
   (2)^2*y(1)+4*x(2)*x(3)*y(1)+x(2)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 5
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 28
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
   2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
   0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
   1)*x(2)*y(0)+6*x(2)^2*y(0)-x(2)^2-4*x(2)*y(0)+x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
```

```
↦ _[1]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[2]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[3]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
  2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
  0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
  1)*x(2)*y(0)+6*x(2)^2*y(0)-x(2)^2-4*x(2)*y(0)+x(2)+y(0)
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 6
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 29
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2+x(2)^2*y(1)-x(2)*y(1)+\
  4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[2]=x(3)^3*y(1)^2
↦ _[3]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2+x(2)^2*y(1)-x(2)*y(1)+\
  4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 7
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 30
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
```

```
↦ _[1]=x(1)^4*y(0)+x(2)^2-x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
  *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[2]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[3]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*y(0)+x(2)^2-x(2)
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 8
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 31
↦ +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
  (2)^2*y(1)+4*x(2)*x(3)*y(1)-x(2)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2
↦ _[2]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ ------------------------- Upcoming Center ---------------------
↦ _[1]=x(2)^4*x(3)^4*y(1)^4+4*x(2)^3*x(3)^3*y(1)^3+6*x(2)^2*x(3)^2*y(1)^2+x\
  (2)^2*y(1)+4*x(2)*x(3)*y(1)-x(2)*y(1)+1
↦ ---------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 9
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 32
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
   2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
   0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
   1)*x(2)*y(0)+6*x(2)^2*y(0)+x(2)^2-4*x(2)*y(0)-x(2)+y(0)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[2]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦ _[3]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
   *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*x(2)^4*y(0)-4*x(1)^3*x(2)^4*y(0)+4*x(1)^3*x(2)^3*y(0)+6*x(1)^\
   2*x(2)^4*y(0)-12*x(1)^2*x(2)^3*y(0)-4*x(1)*x(2)^4*y(0)+6*x(1)^2*x(2)^2*y(\
   0)+12*x(1)*x(2)^3*y(0)+x(2)^4*y(0)-12*x(1)*x(2)^2*y(0)-4*x(2)^3*y(0)+4*x(\
   1)*x(2)*y(0)+6*x(2)^2*y(0)+x(2)^2-4*x(2)*y(0)-x(2)+y(0)
↦ --------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart ++++++++++++++++++++++
↦ Current number of final charts: 10
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 33
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2-x(2)^2*y(1)+x(2)*y(1)+\
   4*x(3)*y(1)+1
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=1
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=y(1)
↦ [4]:
↦    _[1]=x(3)
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(3)^3*y(1)^2
↦ _[2]=x(2)*x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦ _[3]=x(3)^4*y(1)^3+x(3)^3*y(1)^2
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(3)^4*y(1)^4+4*x(3)^3*y(1)^3+6*x(3)^2*y(1)^2-x(2)^2*y(1)+x(2)*y(1)+\
   4*x(3)*y(1)+1
↦ ----------------------------------------------------------------
↦ ++++++++++++++ Overview of Current Chart +++++++++++++++++++++++
↦ Current number of final charts: 11
↦ Total number of charts currently in chart-tree: 34
↦ Index of the current chart in chart-tree: 34
↦ ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
↦
↦ ==== Ambient Space:
↦ _[1]=0
↦
↦ ==== Ideal of Variety:
↦ _[1]=x(1)^4*y(0)-x(2)^2+x(2)
↦
↦ ==== Exceptional Divisors:
↦ [1]:
↦    _[1]=y(0)
↦ [2]:
↦    _[1]=1
↦ [3]:
↦    _[1]=1
↦ [4]:
↦    _[1]=x(1)-1
↦
↦ ==== Images of variables of original ring:
↦ _[1]=x(1)^3*y(0)-3*x(1)^2*y(0)+3*x(1)*y(0)-y(0)
↦ _[2]=x(1)^4*x(2)*y(0)-4*x(1)^3*x(2)*y(0)+x(1)^3*y(0)+6*x(1)^2*x(2)*y(0)-3\
   *x(1)^2*y(0)-4*x(1)*x(2)*y(0)+3*x(1)*y(0)+x(2)*y(0)-y(0)
↦ _[3]=x(1)^4*y(0)-3*x(1)^3*y(0)+3*x(1)^2*y(0)-x(1)*y(0)
↦
↦ -------------------------- Upcoming Center ---------------------
↦ _[1]=x(1)^4*y(0)-x(2)^2+x(2)
↦ ----------------------------------------------------------------
↦ ============= result will be tested ==========
↦
↦ the number of charts obtained: 12
↦ =============     result is o.k.    ==========
list iden=prepEmbDiv(re);
intvec v=computeN(re,iden);
```

```
v;
↦ 3,4,8,12,1
```

D.5.17 schubert_lib

Library: schubert.lib

Purpose: Procedures for Intersection Theory

Author: Hiep Dang, email: hiep@mathematik.uni-kl.de

Overview: We implement new classes (variety, sheaf, stack, graph) and methods for computing with them. An abstract variety is represented by a nonnegative integer which is its dimension and a graded ring which is its Chow ring. An abstract sheaf is represented by a variety and a polynomial which is its Chern character. In particular, we implement the concrete varieties such as projective spaces, Grassmannians, and projective bundles.

An important task of this library is related to the computation of Gromov-Witten invariants. In particular, we implement new tools for the computation in equivariant intersection theory. These tools are based on the localization of moduli spaces of stable maps and Bott's formula. They are useful for the computation of Gromov-Witten invariants. In order to do this, we have to deal with moduli spaces of stable maps, which were introduced by Kontsevich, and the graphs corresponding to the fixed point components of a torus action on the moduli spaces of stable maps.

As an insightful example, the numbers of rational curves on general complete intersection Calabi-Yau threefolds in projective spaces are computed up to degree 6. The results are all in agreement with predictions made from mirror symmetry computations.

References:

Hiep Dang, Intersection theory with applications to the computation of Gromov-Witten invariants, Ph.D thesis, TU Kaiserslautern, 2013.

Sheldon Katz and Stein A. Stromme, Schubert-A Maple package for intersection theory and enumerative geometry, 1992.

Daniel R. Grayson, Michael E. Stillman, Stein A. Stromme, David Eisenbud and Charley Crissman, Schubert2-A Macaulay2 package for computation in intersection theory, 2010.

Maxim Kontsevich, Enumeration of rational curves via torus actions, 1995.

Procedures:

D.5.17.1 makeVariety

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: makeVariety(d,i); d int, i ideal

Assume: d is a nonnegative integer, i is an ideal

Return: variety

Theory: create an abstract variety which has dimension d, and its Chow ring should be a quotient ring

Example:

```
LIB "schubert.lib";
ring r = 0,(h,e),wp(1,1);
ideal rels = he,h2+e2;
variety V = makeVariety(2,rels);
V;
↦ A variety of dimension 2
↦
V.dimension;
↦ 2
V.relations;
↦ _[1]=he
↦ _[2]=h2+e2
```

D.5.17.2 printVariety

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: printVariety(V); V variety

Assume: V is an abstract variety

Theory: This is the print function used by Singular to print an abstract variety.

Example:

```
LIB "schubert.lib";
ring r = 0,(h,e),wp(1,1);
ideal rels = he,h2+e2;
variety V = makeVariety(2,rels);
V;
↦ A variety of dimension 2
↦
```

D.5.17.3 productVariety

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: productVariety(U,V); U variety, V variety

Input: two abstract varieties

Output: a product variety as an abstract variety

Return: variety

Example:

```
LIB "schubert.lib";
variety P = projectiveSpace(3);
variety G = Grassmannian(2,4);
variety W = productVariety(P,G);
W;
↦ A variety of dimension 7
↦
W.dimension == P.dimension + G.dimension;
↦ 1
def r = W.baseRing;
setring r;
W.relations;
```

```
↦ _[1]=h^4
↦ _[2]=q(1)^3-2*q(1)*q(2)
↦ _[3]=q(1)^4-3*q(1)^2*q(2)+q(2)^2
```

See also: Section D.5.17.5 [Grassmannian], page 1588; Section D.5.17.7 [projectiveBundle], page 1590; Section D.5.17.6 [projectiveSpace], page 1589.

D.5.17.4 ChowRing

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: ChowRing(V); V variety

Assume: V is an abstract variety

Return: qring

Example:

```
LIB "schubert.lib";
ring r = 0,(h,e),wp(1,1);
ideal rels = he,h2+e2;
int d = 2;
variety V = makeVariety(2,rels);
ChowRing(V);
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    h e
↦ //        block   2 : ordering C
↦ // quotient ring from ideal ...
```

D.5.17.5 Grassmannian

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: Grassmannian(k,n); k int, n int

Return: variety

Theory: create a Grassmannian G(k,n) as an abstract variety. This abstract variety has diemnsion k(n-k) and its Chow ring is the quotient ring of a polynomial ring in n-k variables q(1),...,q(n-k), which are the Chern classes of tautological quotient bundle on G(k,n), modulo some ideal generated by n-k polynomials which come from the Giambelli formula. The monomial ordering of this Chow ring is 'wp' with vector (1..k,1..n-k). Moreover, we export the Chern characters of tautological subbundle and quotient bundle on G(k,n)
(say 'subBundle' and 'quotientBundle').

Example:

```
LIB "schubert.lib";
variety G24 = Grassmannian(2,4);
G24;
↦ A variety of dimension 4
↦
def r = G24.baseRing;
setring r;
subBundle;
```

```
↦ 1/6*q(1)*q(2)-1/2*q(1)^2+q(2)-q(1)+2
quotientBundle;
↦ -1/6*q(1)*q(2)+1/2*q(1)^2-q(2)+q(1)+2
G24.dimension;
↦ 4
G24.relations;
↦ _[1]=q(1)^3-2*q(1)*q(2)
↦ _[2]=q(1)^4-3*q(1)^2*q(2)+q(2)^2
ChowRing(G24);
↦ // coefficients: QQ
↦ // number of vars : 2
↦ //        block   1 : ordering wp
↦ //                  : names    q(1) q(2)
↦ //                  : weights     1    2
↦ //        block   2 : ordering C
↦ // quotient ring from ideal ...
```

See also: Section D.5.17.7 [projectiveBundle], page 1590; Section D.5.17.6 [projectiveSpace], page 1589.

D.5.17.6 projectiveSpace

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: projectiveSpace(n); n int

Return: variety

Theory: create a projective space of dimension n as an abstract variety. Its Chow ring is a quotient ring in one variable h modulo the ideal generated by h^(n+1).

Example:

```
LIB "schubert.lib";
variety P = projectiveSpace(3);
P;
↦ A variety of dimension 3
↦
P.dimension;
↦ 3
def r = P.baseRing;
setring r;
P.relations;
↦ _[1]=h4
ChowRing(P);
↦ // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering wp
↦ //                  : names    h
↦ //                  : weights  1
↦ //        block   2 : ordering C
↦ // quotient ring from ideal ...
```

See also: Section D.5.17.5 [Grassmannian], page 1588; Section D.5.17.7 [projectiveBundle], page 1590.

D.5.17.7 projectiveBundle

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: projectiveBundle(S); S sheaf

Input: a sheaf on an abstract variety

Return: variety

Theory: create a projective bundle as an abstract variety. This is related to the enumeration of conics.

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(3,5);
def r = G.baseRing;
setring r;
sheaf S = makeSheaf(G,subBundle);
sheaf B = dualSheaf(S)^2;
variety PB = projectiveBundle(B);
PB;
↦ A variety of dimension 11
↦
def R = PB.baseRing;
setring R;
QuotientBundle;
↦ 1/1995840*z^5*q(2)^3-1/120960*z^5*q(1)*q(2)^2+1/20160*z^5*q(1)^2*q(2)+1/4\
   0320*z^5*q(2)^2-1/10080*z^5*q(1)^3-1/2688*z^5*q(1)*q(2)+1/840*z^5*q(1)^2+\
   1/1008*z^5*q(2)-1/180*z^5*q(1)+1/120*z^5-1/36288*z^4*q(2)^3+5/12096*z^4*q\
   (1)*q(2)^2-1/448*z^4*q(1)^2*q(2)+5/8064*z^4*q(2)^2+1/252*z^4*q(1)^3+1/100\
   8*z^4*q(1)*q(2)-1/72*z^4*q(1)^2+1/144*z^4*q(2)+1/24*z^4+43/72576*z^3*q(2)\
   ^3-47/8064*z^3*q(1)*q(2)^2+11/504*z^3*q(1)^2*q(2)-1/84*z^3*q(2)^2-1/36*z^\
   3*q(1)^3+5/144*z^3*q(1)*q(2)+1/6*z^3-1/192*z^2*q(2)^3+1/36*z^2*q(1)*q(2)^\
   2-1/24*z^2*q(1)^2*q(2)+1/36*z^2*q(2)^2+1/2*z^2+1/63*z*q(2)^3-1/36*z*q(1)*\
   q(2)^2+z+1
ChowRing(PB);
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering wp
↦ //                  : names    z
↦ //                  : weights  1
↦ //        block   2 : ordering wp
↦ //                  : names    q(1) q(2)
↦ //                  : weights     1    2
↦ //        block   3 : ordering C
↦ // quotient ring from ideal ...
```

See also: Section D.5.17.5 [Grassmannian], page 1588; Section D.5.17.6 [projectiveSpace], page 1589.

D.5.17.8 integral

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: integral(V,f); V variety, f poly

Input: a abstract variety and a polynomial

Return: int

Purpose: computing intersection numbers.

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def r = G.baseRing;
setring r;
integral(G,q(1)^4);
↦ 2
```

D.5.17.9 makeSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: makeSheaf(V,ch); V variety, ch poly

Return: sheaf

Theory: create a sheaf on an abstract variety, and its Chern character is the polynomial ch.

Example:

```
LIB "schubert.lib";
variety X;
X.dimension = 4;
ring r = 0,(c(1..2),d(1..3)),wp(1..2,1..3);
setring r;
X.baseRing = r;
poly c = 1 + c(1) + c(2);
poly ch = 2 + logg(c,4);
sheaf S = makeSheaf(X,ch);
S;
↦ A sheaf of rank  2
↦
```

See also: Section D.5.17.10 [printSheaf], page 1591; Section D.5.17.11 [rankSheaf], page 1592.

D.5.17.10 printSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: printSheaf(S); S sheaf

Return: string

Input: a sheaf

Theory: This is the print function used by Singular to print a sheaf.

Example:

```
LIB "schubert.lib";
variety X;
X.dimension = 4;
ring r = 0,(c(1..2),d(1..3)),wp(1..2,1..3);
setring r;
X.baseRing = r;
```

```
poly c = 1 + c(1) + c(2);
poly ch = 2 + logg(c,4);
sheaf S = makeSheaf(X,ch);
S;
↦ A sheaf of rank  2
↦
```

See also: Section D.5.17.9 [makeSheaf], page 1591; Section D.5.17.11 [rankSheaf], page 1592.

D.5.17.11 rankSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: rankSheaf(S); S sheaf

Return: int

Input: S is a sheaf

Output: a positive integer which is the rank of a sheaf.

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,subBundle);
rankSheaf(S);
↦ 2
```

See also: Section D.5.17.9 [makeSheaf], page 1591; Section D.5.17.10 [printSheaf], page 1591.

D.5.17.12 totalChernClass

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: totalChernClass(S); S sheaf

Return: poly

Input: S is a sheaf

Output: a polynomial which is the total Chern class of a sheaf

Example:

```
LIB "schubert.lib";
variety X;
X.dimension = 4;
ring r = 0,(c(1..2),d(1..3)),wp(1..2,1..3);
setring r;
X.baseRing = r;
poly c = 1 + c(1) + c(2);
poly ch = 2 + logg(c,4);
sheaf E = makeSheaf(X,ch);
sheaf S = E^3;
totalChernClass(S);
↦ 18*c(1)^2*c(2)+9*c(2)^2+6*c(1)^3+30*c(1)*c(2)+11*c(1)^2+10*c(2)+6*c(1)+1
```

See also: Section D.5.17.13 [ChernClass], page 1593; Section D.5.17.14 [topChernClass], page 1593; Section D.5.17.15 [totalSegreClass], page 1594.

D.5.17.13 ChernClass

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: ChernClass(S,i); S sheaf, i int

Input: S is a sheaf, i is a nonnegative integer

Return: poly

Theory: This is the i-th Chern class of a sheaf

Example:

```
LIB "schubert.lib";
variety X;
X.dimension = 4;
ring r = 0,(c(1..2),d(1..3)),wp(1..2,1..3);
setring r;
X.baseRing = r;
poly c = 1 + c(1) + c(2);
poly ch = 2 + logg(c,4);
sheaf E = makeSheaf(X,ch);
sheaf S = E^3;
ChernClass(S,1);
↦ 6*c(1)
ChernClass(S,2);
↦ 11*c(1)^2+10*c(2)
ChernClass(S,3);
↦ 6*c(1)^3+30*c(1)*c(2)
ChernClass(S,4);
↦ 18*c(1)^2*c(2)+9*c(2)^2
```

See also: Section D.5.17.14 [topChernClass], page 1593; Section D.5.17.12 [totalChernClass], page 1592.

D.5.17.14 topChernClass

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: topChernClass(S); S sheaf

Return: poly

Input: S is a sheaf

Theory: This is the top Chern class of a sheaf

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,quotientBundle);
sheaf B = S^3;
topChernClass(B);
↦ 27*q(2)^2
```

See also: Section D.5.17.13 [ChernClass], page 1593; Section D.5.17.12 [totalChernClass], page 1592.

D.5.17.15 totalSegreClass

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: totalSegreClass(S); S sheaf

Return: poly

Input: S is a sheaf

Theory: This is the total Segre class of a sheaf.
SEE AlSO: totalChernClass

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,subBundle);
totalSegreClass(S);
↦ q(2)+q(1)+1
```

D.5.17.16 dualSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: dualSheaf(S); S sheaf

Return: sheaf

Theory: This is the dual of a sheaf

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,subBundle);
sheaf D = dualSheaf(S);
D;
↦ A sheaf of rank  2
↦
```

See also: Section D.5.17.20 [addSheaf], page 1596; Section D.5.17.19 [quotSheaf], page 1595; Section D.5.17.18 [symmetricPowerSheaf], page 1595; Section D.5.17.17 [tensorSheaf], page 1594.

D.5.17.17 tensorSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: tensorSheaf(A,B); A sheaf, B sheaf

Return: sheaf

Theory: This is the tensor product of two sheaves

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(3,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,subBundle);
sheaf Q = makeSheaf(G,quotientBundle);
sheaf T = S*Q;
T;
↦ A sheaf of rank  3
↦
```

See also: Section D.5.17.20 [addSheaf], page 1596; Section D.5.17.16 [dualSheaf], page 1594; Section D.5.17.19 [quotSheaf], page 1595; Section D.5.17.18 [symmetricPowerSheaf], page 1595.

D.5.17.18 symmetricPowerSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: symmetricPowerSheaf(S,n); S sheaf, n int

Return: sheaf

Theory: This is the n-th symmetric power of a sheaf

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def R = G.baseRing;
setring R;
sheaf S = makeSheaf(G,quotientBundle);
sheaf B = symmetricPowerSheaf(S,3);
B;
↦ A sheaf of rank  4
↦
sheaf A = S^3;
A;
↦ A sheaf of rank  4
↦
A.ChernCharacter == B.ChernCharacter;
↦ 1
```

See also: Section D.5.17.20 [addSheaf], page 1596; Section D.5.17.16 [dualSheaf], page 1594; Section D.5.17.19 [quotSheaf], page 1595; Section D.5.17.17 [tensorSheaf], page 1594.

D.5.17.19 quotSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: quotSheaf(A,B); A sheaf, B sheaf

Return: sheaf

Theory: This is the quotient of two sheaves

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(3,5);
def r = G.baseRing;
```

```
setring r;
sheaf S = makeSheaf(G,subBundle);
sheaf B = dualSheaf(S)^2;
sheaf B3 = dualSheaf(S)^3;
sheaf B5 = dualSheaf(S)^5;
variety PB = projectiveBundle(B);
def R = PB.baseRing;
setring R;
sheaf Q = makeSheaf(PB,QuotientBundle);
sheaf V = dualSheaf(Q)*B3;
sheaf A = B5 - V;
A;
↦ A sheaf of rank  11
↦
```

See also: Section D.5.17.20 [addSheaf], page 1596; Section D.5.17.16 [dualSheaf], page 1594; Section D.5.17.18 [symmetricPowerSheaf], page 1595; Section D.5.17.17 [tensorSheaf], page 1594.

D.5.17.20 addSheaf

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: addSheaf(A,B); A sheaf, B sheaf

Return: sheaf

Theory: This is the direct sum of two sheaves.

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(3,5);
def r = G.baseRing;
setring r;
sheaf S = makeSheaf(G,subBundle);
sheaf Q = makeSheaf(G,quotientBundle);
sheaf D = S + Q;
D;
↦ A sheaf of rank  5
↦
D.ChernCharacter == rankSheaf(D);
↦ 1
totalChernClass(D) == 1;
↦ 1
```

See also: Section D.5.17.16 [dualSheaf], page 1594; Section D.5.17.19 [quotSheaf], page 1595; Section D.5.17.18 [symmetricPowerSheaf], page 1595; Section D.5.17.17 [tensorSheaf], page 1594.

D.5.17.21 makeGraphVE

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Assume: v is a list of vertices, e is a list of edges.

Return: graph with vertices v and edges e.

Theory: Creates a graph from a list of vertices and edges.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
graph G = makeGraphVE(list(list(0,1,list(0,1,2)),list(1,1,list(1,0,2))),
list(list(0,1,2)));
G;
↦ A graph with 2 vertices and 1 edges
↦
```

D.5.17.22 printGraphG

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: printGraphG(G); G graph

Assume: G is a graph.

Theory: This is the print function used by Singular to print a graph.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
graph G = makeGraphVE(list(list(0,1,list(0,1,2)),list(1,1,list(1,0,2))),
list(list(0,1,2)));
G;
↦ A graph with 2 vertices and 1 edges
↦
```

D.5.17.23 moduliSpace

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: moduliSpace(V,d); V variety, d int

Assume: V is a projective space and d is a positive integer.

Theory: This is the function used by Singular to create a moduli space of stable maps from a genus zero curve to a projective space.

Example:

```
LIB "schubert.lib";
ring r = 0,(x),dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
M;
↦ A moduli space of dimension 11
↦
```

D.5.17.24 printStack

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: printStack(M); M stack

Assume: M is a moduli space of stable maps.

Theory: This is the print function used by Singular to print a stack.

Example:

```
LIB "schubert.lib";
ring r = 0,(x),dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
M;
↦ A moduli space of dimension 11
↦
```

D.5.17.25 dimStack

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: dimStack(M); M stack

Return: int

Input: M is a moduli space of stable maps.

Output: the dimension of moduli space of stable maps.

Example:

```
LIB "schubert.lib";
ring r = 0,(x),dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
dimStack(M);
↦ 11
```

D.5.17.26 fixedPoints

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: fixedPoints(M); M stack

Return: list

Input: M is a moduli space of stable maps.

Output: a list of graphs corresponding the fixed point components of a torus action on a moduli space of stable maps.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
def F = fixedPoints(M);
size(F);
↦ 100
typeof(F[1]) == "list";
↦ 1
typeof(F[1][1]) == "graph";
↦ 1
typeof(F[1][2]) == "int";
↦ 1
```

D.5.17.27 contributionBundle

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: contributionBundle(M,G,#); M stack, G graph, # list

Return: number

Input: M is a moduli space of stable maps, G is a graph, # is a list.

Output: a number corresponding to the contribution bundle on a moduli space of stable maps at a fixed point component (graph)

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
def F = fixedPoints(M);
graph G = F[1][1];
number f = contributionBundle(M,G);
number g = contributionBundle(M,G,5);
f == g;
↦ 1
```

See also: Section D.5.17.28 [normalBundle], page 1599.

D.5.17.28 normalBundle

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Return: number

Input: M is a moduli space of stable maps, G is a graph

Output: a number corresponding to the normal bundle on a moduli space of stable maps at a graph

hypersurfaces

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
def F = fixedPoints(M);
graph G = F[1][1];
number f = normalBundle(M,G);
f <> 0;
↦ 1
```

See also: Section D.5.17.27 [contributionBundle], page 1599.

D.5.17.29 multipleCover

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: multipleCover(d); d int

Return: number

Theory: This is the contribution of degree d multiple covers of a smooth rational curve as a Gromov-Witten invariant.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
multipleCover(1);
↦ 1
multipleCover(2);
↦ 1/8
multipleCover(3);
↦ 1/27
multipleCover(4);
↦ 1/64
multipleCover(5);
↦ 1/125
multipleCover(6);
↦ 1/216
```

See also: Section D.5.17.30 [linesHypersurface], page 1600; Section D.5.17.31 [rationalCurve], page 1601.

D.5.17.30 linesHypersurface

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: linesHypersurface(n); n int

Return: number

Theory: This is the number of lines on a general hypersurface of degree d = 2n-3 in an n-dimensional projective space.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
linesHypersurface(2);
↦ 1
linesHypersurface(3);
↦ 27
linesHypersurface(4);
↦ 2875
linesHypersurface(5);
↦ 698005
linesHypersurface(6);
↦ 305093061
linesHypersurface(7);
↦ 210480374951
linesHypersurface(8);
↦ 210776836330775
linesHypersurface(9);
↦ 289139638632755625
linesHypersurface(10);
↦ 520764738758073845321
```

See also: Section D.5.17.30 [linesHypersurface], page 1600; Section D.5.17.29 [multipleCover], page 1599.

D.5.17.31 rationalCurve

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: rationalCurve(d,#); d int, # list

Return: number

Theory: This is the Gromov-Witten invariant corresponding the number of rational curves on a general Calabi-Yau threefold.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
rationalCurve(1);
↦ 2875
/*
rationalCurve(2);
rationalCurve(3);
rationalCurve(4);
rationalCurve(1,list(4,2));
rationalCurve(1,list(3,3));
rationalCurve(1,list(3,2,2));
rationalCurve(1,list(2,2,2,2));
rationalCurve(2,list(4,2));
rationalCurve(2,list(3,3));
rationalCurve(2,list(3,2,2));
rationalCurve(2,list(2,2,2,2));
rationalCurve(3,list(4,2));
rationalCurve(3,list(3,3));
rationalCurve(3,list(3,2,2));
rationalCurve(3,list(2,2,2,2));
rationalCurve(4,list(4,2));
rationalCurve(4,list(3,3));
rationalCurve(4,list(3,2,2));
rationalCurve(4,list(2,2,2,2));
*/
```

See also: Section D.5.17.30 [linesHypersurface], page 1600; Section D.5.17.29 [multipleCover], page 1599.

D.5.17.32 sumofquotients

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: sumofquotient(M,F,#); M stack, F list, # list

Return: number

Theory: This is useful for the parallel computation of rationalCurve.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
variety P = projectiveSpace(4);
stack M = moduliSpace(P,2);
list F = fixedPoints(M);
```

```
sumofquotients(M,F);
↦ 4876875/8
sumofquotients(M,F,list(5));
↦ 4876875/8
```

D.5.17.33 homog_part

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: homog_part(f,n); f poly, n int

Return: poly

Purpose: computing the homogeneous component of a polynomial.

Example:

```
LIB "schubert.lib";
ring r = 0,(x,y,z),wp(1,2,3);
poly f = 1+x+x2+x3+x4+y+y2+y3+z+z2+xy+xz+yz+xyz;
homog_part(f,0);
↦ 1
homog_part(f,1);
↦ x
homog_part(f,2);
↦ x2+y
homog_part(f,3);
↦ x3+xy+z
homog_part(f,4);
↦ x4+y2+xz
homog_part(f,5);
↦ yz
homog_part(f,6);
↦ y3+xyz+z2
```

D.5.17.34 homog_parts

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: homog_parts(f,i,j); f poly, i int, j int

Return: poly

Theory: computing a polynomial which is the sum of the homogeneous components of a polynomial.

Example:

```
LIB "schubert.lib";
ring r = 0,(x,y,z),wp(1,2,3);
poly f = 1+x+x2+x3+x4+y+y2+y3+z+z2+xy+xz+yz+xyz;
homog_parts(f,2,4);
↦ x4+y2+xz+x3+xy+z+x2+y
```

D.5.17.35 logg

Procedure from library `schubert.lib` (see Section D.5.17 [schubert_lib], page 1586).

Usage: logg(f,n); f poly, n int

Return: poly

Theory: computing Chern characters from total Chern classes.

Example:

```
LIB "schubert.lib";
ring r = 0,(x,y),wp(1,2);
poly f = 1+x+y;
logg(f,4);
↦ 1/24x4-1/6x2y+1/12y2+1/6x3-1/2xy+1/2x2-y+x
```

D.5.17.36 expp

Procedure from library schubert.lib (see Section D.5.17 [schubert_lib], page 1586).

Usage: expp(f,n); f poly, n int

Return: poly

Purpose: computing total Chern classes from Chern characters.

Example:

```
LIB "schubert.lib";
ring r = 0,x,dp;
poly f = 3+x;
expp(f,3);
↦ 1/6x3+1/2x2+x+1
```

D.5.17.37 SchubertClass

Procedure from library schubert.lib (see Section D.5.17 [schubert_lib], page 1586).

Usage: SchubertClass(p); p list

Input: a list of integers which is a partition

Return: poly

Purpose: compute the Schubert classes on a Grassmannian.

Example:

```
LIB "schubert.lib";
variety G = Grassmannian(2,4);
def r = G.baseRing;
setring r;
list p = 1,1;
SchubertClass(p);
↦ q(1)^2-q(2)
```

D.5.17.38 dualPartition

Procedure from library schubert.lib (see Section D.5.17 [schubert_lib], page 1586).

Usage: dualPartition(k,n,p); k int, n int, p list

Input: two integers and a partition

Return: list

Purpose: compute the dual of a partition.

Example:

```
LIB "schubert.lib";
ring r = 0,(x),dp;
dualPartition(2,4,list(2,1));
↦ [1]:
↦    1
↦ [2]:
↦    0
```

See also: Section D.5.17.37 [SchubertClass], page 1603.

D.5.18 sheafcoh_lib

Library: sheafcoh.lib

Purpose: Procedures for Computing Sheaf Cohomology

Authors: Wolfram Decker, decker@mathematik.uni-kl.de
Christoph Lossen, lossen@mathematik.uni-kl.de
Gerhard Pfister, pfister@mathematik.uni-kl.de
Oleksandr Motsak, U@D, where U={motsak}, D={mathematik.uni-kl.de}

Procedures:

D.5.18.1 truncate

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: truncate(M,d); M module, d int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute

Return: module

Note: Output is a presentation matrix for the truncation of coker(M) at degree d.

Example:

```
LIB "sheafcoh.lib";
ring R=0,(x,y,z),dp;
module M=maxideal(3);
homog(M);
↦ 1
// compute presentation matrix for truncated module (R/<x,y,z>^3)_(>=2)
module M2=truncate(M,2);
print(M2);
↦ z,0,0,0,0,0,y, 0, 0,0, 0,0,x, 0, 0, 0, 0, 0,
↦ 0,z,0,0,0,0,-z,y, 0,0, 0,0,0, x, 0, 0, 0, 0,
↦ 0,0,z,0,0,0,0, -z,y,0, 0,0,0, 0, x, 0, 0, 0,
↦ 0,0,0,z,0,0,0, 0, 0,y, 0,0,-z,-y,0, x, 0, 0,
↦ 0,0,0,0,z,0,0, 0, 0,-z,y,0,0, 0, -y,0, x, 0,
↦ 0,0,0,0,0,z,0, 0, 0,0, 0,y,0, 0, 0, -z,-y,x
dimGradedPart(M2,1);
↦ 0
dimGradedPart(M2,2);
↦ 6
// this should coincide with:
dimGradedPart(M,2);
```

```
↦ 6
// shift grading by 1:
intvec v=1;
attrib(M,"isHomog",v);
M2=truncate(M,2);
print(M2);
↦ 0, -y,-z,z2,0, 0, yz,0, xz,y2,0, xy,x2,
↦ -z,x, 0, 0, z2,0, 0, yz,0, 0, y2,0, 0,
↦ y, 0, x, 0, 0, z2,0, 0, 0, 0, 0, 0, 0
dimGradedPart(M2,3);
↦ 6
```

D.5.18.2 truncateFast

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: truncateFast(M,d); M module, d int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute 'isHomog'

Return: module

Note: Output is a presentation matrix for the truncation of coker(M) at d.
Fast + experimental version. M should be a SB!

Display: If `printlevel`>=1, step-by step timings will be printed. If `printlevel`>=2 we add progress debug messages if `printlevel`>=3, even all intermediate results...

Example:

```
LIB "sheafcoh.lib";
ring R=0,(x,y,z,u,v),dp;
module M=maxideal(3);
homog(M);
↦ 1
// compute presentation matrix for truncated module (R/<x,y,z,u>^3)_(>=2)
int t=timer;
module M2t=truncate(M,2);
t = timer - t;
"// Simple truncate: ", t;
↦ // Simple truncate:  0
t=timer;
module M2=truncateFast(std(M),2);
t = timer - t;
"// Fast truncate: ", t;
↦ // Fast truncate:  0
print(M2);
↦ v,0,0,0,0,0,0,0,0,0,0,0,0,0,0,u, 0, 0, 0, 0, 0,0,0,0,0,0,0,0,0,0,z, 0, 0,\
    0, 0,
↦   0, 0, 0, 0, 0,0,0,0,0,0,y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,0,x, 0,\
    0, 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,v,0,0,0,0,0,0,0,0,0,0,0,0,0,-v,u, 0, 0, 0, 0,0,0,0,0,0,0,0,0,0,0, z, 0,\
    0, 0,
↦   0, 0, 0, 0, 0,0,0,0,0,0,0, y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,0,0, x,\
    0, 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,v,0,0,0,0,0,0,0,0,0,0,0,0,0, 0, u, 0, 0, 0,0,0,0,0,0,0,0,0,0,-v,-u,z,\
    0, 0,
↦   0, 0, 0, 0, 0,0,0,0,0,0,0, 0, y, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,0,0, 0,\
    x, 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,v,0,0,0,0,0,0,0,0,0,0,0,0, 0, 0, u, 0, 0,0,0,0,0,0,0,0,0,0,0, 0, 0,\
    z, 0,
↦   0, 0, 0, 0, 0,0,0,0,0,0,-v,-u,-z,y, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,0,0, 0,\
    0, x,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,v,0,0,0,0,0,0,0,0,0,0,0, 0, 0, 0, u, 0,0,0,0,0,0,0,0,0,0,0, 0, 0,\
    0, z,
↦   0, 0, 0, 0, 0,0,0,0,0,0,0, 0, 0, 0, y, 0, 0, 0, 0, 0, 0, 0, 0,0,0,-v,-u\
   ,-z,-y,
↦   x, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,v,0,0,0,0,0,0,0,0,0,0, -v,0, 0, 0, u,0,0,0,0,0,0,0,0,0,0, 0, 0,\
    0, 0,
↦   z, 0, 0, 0, 0,0,0,0,0,0,0, 0, 0, 0, 0, y, 0, 0, 0, 0, 0, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, x, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,v,0,0,0,0,0,0,0,0,0, 0, -v,0, 0, 0,u,0,0,0,0,0,0,0,0,0, 0, 0,\
    0, 0,
↦   -u,z, 0, 0, 0,0,0,0,0,0,0, 0, 0, 0, 0, 0, y, 0, 0, 0, 0, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, 0, x, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,v,0,0,0,0,0,0,0,0, 0, 0, -v,0, 0,0,u,0,0,0,0,0,0,0,0, 0, 0,\
    0, 0,
↦   0, 0, z, 0, 0,0,0,0,0,0,0, 0, 0, 0, 0, -u,-z,y, 0, 0, 0, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, 0, 0, x, 0, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,v,0,0,0,0,0,0,0, 0, 0, 0, -v,0,0,0,u,0,0,0,0,0,0,0, 0, 0,\
    0, 0,
↦   0, 0, 0, z, 0,0,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, y, 0, 0, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, -u,-z,-y,x, 0, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,v,0,0,0,0,0,0, 0, 0, 0, 0, 0,0,0,0,u,0,0,0,0,0,0, 0, -v\
   ,0, 0,
↦   0, -u,0, 0, z,0,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, y, 0, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, 0, 0, 0, 0, x, 0, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,0,v,0,0,0,0,0, 0, 0, 0, 0, 0,0,0,0,0,u,0,0,0,0,0, 0, 0,\
    -v,0,
↦   0, 0, -u,0, 0,z,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, -z,y, 0, 0,0,0,0, 0,\
    0, 0,
↦   0, 0, 0, 0, 0, 0, x, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,v,0,0,0,0, 0, 0, 0, 0, 0,0,0,0,0,0,u,0,0,0,0, 0, 0,\
    0, -v,
↦   0, 0, 0, -u,0,0,z,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, y, 0,0,0,0, 0,\
    0, 0,
↦   0, 0, 0, 0, 0, -z,-y,x, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,v,0,0,0, 0, 0, 0, 0, 0,0,0,0,0,0,0,u,0,0,0, 0, 0,\
    0, 0,
↦   0, 0, 0, 0, 0,0,0,z,0,0,0, 0, 0, -v,0, 0, 0, -u,0, 0, -z,0, y,0,0,0, 0,\
   0, 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, x, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,v,0,0, 0, 0, 0, 0, 0,0,0,0,0,0,0,0,u,0,0, 0, 0,\
   0, 0,
↦   0, 0, 0, 0, 0,0,0,0,z,0,0, 0, 0, 0, -v,0, 0, 0, -u,0, 0, -z,0,y,0,0, 0,\
   0, 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, -y,x, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,v,0, 0, 0, 0, 0, 0,0,0,0,0,0,0,0,0,u,0, 0, 0,\
   0, 0,
↦   0, 0, 0, 0, 0,0,0,0,0,z,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0,y,0, 0,\
   0, 0,
↦   -v,0, 0, 0, -u,0, 0, -z,0, -y,x
"// Check: M2t == M2?: ", size(NF(M2, std(M2t))) + size(NF(M2t, std(M2)));
↦ // Check: M2t == M2?:  0
dimGradedPart(M2,1);
↦ 0
dimGradedPart(M2,2);
↦ 15
// this should coincide with:
dimGradedPart(M,2);
↦ 15
// shift grading by 1:
intvec v=1;
attrib(M,"isHomog",v);
t=timer;
M2t=truncate(M,2);
t = timer - t;
"// Simple truncate: ", t;
↦ // Simple truncate:  0
t=timer;
M2=truncateFast(std(M),2);
t = timer - t;
"// Fast truncate: ", t;
↦ // Fast truncate:  0
print(M2);
↦ u, z, 0, y, 0, 0, x, 0, 0, 0, v2,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\
  , 0,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ -v,0, z, 0, y, 0, 0, x, 0, 0, 0, v2,0, 0, 0, uv,0, 0, 0, 0, 0, 0, 0, 0, 0\
  , u2,
↦   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, -v,-u,0, 0, y, 0, 0, x, 0, 0, 0, v2,0, 0, 0, uv,0, 0, zv,0, 0, 0, 0, 0\
  , 0,
↦   u2,0, 0, zu,0, 0, 0, 0, 0, z2,0, 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, -v,-u,-z,0, 0, 0, x, 0, 0, 0, v2,0, 0, 0, uv,0, 0, zv,0, yv,0, 0\
  , 0,
↦   0, u2,0, 0, zu,0, yu,0, 0, 0, z2,0, yz,0, 0, y2,0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, -v,-u,-z,-y,0, 0, 0, 0, v2,0, 0, 0, uv,0, 0, zv,0, yv,x\
  v,0,
↦   0, 0, u2,0, 0, zu,0, yu,xu,0, 0, z2,0, yz,xz,0, y2,xy,x2
"// Check: M2t == M2?: ", size(NF(M2, std(M2t))) + size(NF(M2t, std(M2))); //?
↦ // Check: M2t == M2?:  20
```

```
dimGradedPart(M2,3);
↦ 15
```

D.5.18.3 CM_regularity

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: CM_regularity(M); M module

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute

Return: integer, the Castelnuovo-Mumford regularity of coker(M)

Note: procedure calls mres

Example:

```
LIB "sheafcoh.lib";
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
attrib(M,"isHomog",v);
CM_regularity(M);
↦ 2
```

D.5.18.4 sheafCohBGG

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCohBGG(M,l,h); M module, l,h int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute, `h>=l`, and the basering has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Display: The intmat is displayed in a diagram of the following form: with `displayCohom(A,l,h,nvars(r)-1);`

```
             l          l+1                    h
  ----------------------------------------------
     n:    h^n(F(l))   h^n(F(l+1))   ......   h^n(F(h))
           ...............................................
     1:    h^1(F(l))   h^1(F(l+1))   ......   h^1(F(h))
     0:    h^0(F(l))   h^0(F(l+1))   ......   h^0(F(h))
  ----------------------------------------------
   chi:    chi(F(l))   chi(F(l+1))   ......   chi(F(h))
```

A `'-'` in the diagram refers to a zero entry; a `'*'` refers to a negative entry (= dimension not yet determined). refers to a not computed dimension.

Note: This procedure is based on the Bernstein-Gel'fand-Gel'fand correspondence and on Tate resolution (see [Eisenbud, Floystad, Schreyer: Sheaf cohomology and free resolutions over exterior algebras, Trans AMS 355 (2003)]).
`sheafCohBGG(M,l,h)` does not compute all values in the above table. To determine all values of `h^i(F(d))`, `d=l..h`, use `sheafCohBGG(M,l-n,h+n)`.

Example:

```
LIB "sheafcoh.lib";
// cohomology of structure sheaf on P^4:
//-------------------------------------------
ring r=0,x(1..5),dp;
module M=0;
intmat A=sheafCohBGG(M,-9,4);
A;
↦ 70,35,15,5,1,0,0,0,0,0,-1,-1,-1,-1,
↦ -1,0,0,0,0,0,0,0,0,0,0,-1,-1,-1,
↦ -1,-1,0,0,0,0,0,0,0,0,0,0,-1,-1,
↦ -1,-1,-1,0,0,0,0,0,0,0,0,0,0,-1,
↦ -1,-1,-1,-1,0,0,0,0,0,1,5,15,35,70
displayCohom(A,-9,4,nvars(r)-1);
↦       -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4
↦ ------------------------------------------------------------
↦   4:  70  35  15   5   1   -   -   -   -   -   *   *   *   *
↦   3:   *   -   -   -   -   -   -   -   -   -   -   *   *   *
↦   2:   *   *   -   -   -   -   -   -   -   -   -   -   *   *
↦   1:   *   *   *   -   -   -   -   -   -   -   -   -   -   *
↦   0:   *   *   *   *   -   -   -   -   -   1   5  15  35  70
↦ ------------------------------------------------------------
↦ chi:   *   *   *   *   1   0   0   0   0   1   *   *   *   *
// cohomology of cotangential bundle on P^3:
//-------------------------------------------
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
attrib(M,"isHomog",v);
intmat B=sheafCohBGG(M,-8,4);
B;
↦ 189,120,70,36,15,4,0,0,0,0,-1,-1,-1,
↦ -1,0,0,0,0,0,0,0,0,0,0,-1,-1,
↦ -1,-1,0,0,0,0,0,0,1,0,0,0,-1,
↦ -1,-1,-1,0,0,0,0,0,0,0,6,20,45
displayCohom(B,-8,4,nvars(R)-1);
↦         -8   -7   -6   -5   -4   -3   -2   -1    0    1    2    3    4
↦ ----------------------------------------------------------------------
↦   3:   189  120   70   36   15    4    -    -    -    -    *    *    *
↦   2:     *    -    -    -    -    -    -    -    -    -    -    *    *
↦   1:     *    *    -    -    -    -    -    -    1    -    -    -    *
↦   0:     *    *    *    -    -    -    -    -    -    -    6   20   45
↦ ----------------------------------------------------------------------
↦ chi:     *    *    *  -36  -15   -4    0    0   -1    0    *    *    *
```

See also: Section D.5.18.10 [dimH], page 1618; Section D.5.18.12 [displayCohom], page 1619; Section D.5.18.8 [sheafCoh], page 1615.

D.5.18.5 sheafCohBGGregul

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCohBGGregul(M,l,h); M module, l,h int, reg int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute, `h>=l`, and the basering has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Note: This procedure is based on the Bernstein-Gel'fand-Gel'fand correspondence and on Tate resolution (see [Eisenbud, Floystad, Schreyer: Sheaf cohomology and free resolutions over exterior algebras, Trans AMS 355 (2003)]).
`sheafCohBGG(M,l,h)` does not compute all values in the above table. To determine all values of `h^i(F(d))`, `d=l..h`, use `sheafCohBGG(M,l-n,h+n)`.

Example:

```
LIB "sheafcoh.lib";
// cohomology of structure sheaf on P^4:
//-------------------------------------------
ring r=0,x(1..5),dp;
module M=0;
def A=sheafCohBGGregul(M,-9,4,CM_regularity(M));
A;
↦ 70,35,15,5,1,0,0,0,0,0,-1,-1,-1,-1,
↦ -1,0,0,0,0,0,0,0,0,0,0,-1,-1,-1,
↦ -1,-1,0,0,0,0,0,0,0,0,0,0,-1,-1,
↦ -1,-1,-1,0,0,0,0,0,0,0,0,0,0,-1,
↦ -1,-1,-1,-1,0,0,0,0,0,1,5,15,35,70
// cohomology of cotangential bundle on P^3:
//-------------------------------------------
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
attrib(M,"isHomog",v);
def B=sheafCohBGGregul(M,-8,4,CM_regularity(M));
B;
↦ 189,120,70,36,15,4,0,0,0,0,-1,-1,-1,
↦ -1,0,0,0,0,0,0,0,0,0,0,-1,-1,
↦ -1,-1,0,0,0,0,0,0,1,0,0,0,-1,
↦ -1,-1,-1,0,0,0,0,0,0,0,6,20,45
```

See also: Section D.5.18.10 [dimH], page 1618; Section D.5.18.12 [displayCohom], page 1619; Section D.5.18.8 [sheafCoh], page 1615.

D.5.18.6 sheafCohBGGregul_w

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCohBGGregul_w(M,l,h,w); M module, l,h int, reg int, w intvec

Assume: `M` is graded, and it comes assigned with an admissible degree vector w, `h>=l`, and the basering has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Note: This procedure is based on the Bernstein-Gel'fand-Gel'fand correspondence and on Tate resolution (see [Eisenbud, Floystad, Schreyer: Sheaf cohomology and free resolutions

over exterior algebras, Trans AMS 355 (2003)]).
`sheafCohBGG(M,l,h)` does not compute all values in the above table. To determine all values of `h^i(F(d))`, `d=l..h`, use `sheafCohBGG(M,l-n,h+n)`.

Example:

```
LIB "sheafcoh.lib";
// cohomology of cotangential bundle on P^3:
//-------------------------------------------
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
def B=sheafCohBGGregul_w(M,-8,4,CM_regularity(M),v);
B;
↦ 189,120,70,36,15,4,0,0,0,0,-1,-1,-1,
↦ -1,0,0,0,0,0,0,0,0,0,0,-1,-1,
↦ -1,-1,0,0,0,0,0,1,0,0,0,-1,
↦ -1,-1,-1,0,0,0,0,0,0,0,6,20,45
```

See also: Section D.5.18.10 [dimH], page 1618; Section D.5.18.12 [displayCohom], page 1619; Section D.5.18.8 [sheafCoh], page 1615.

D.5.18.7 sheafCohBGG2

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCohBGG2(M,l,h); M module, l,h int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute, `h>=l`, and the basering has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Display: The intmat is displayed in a diagram of the following form: with `displayCohom(A,l,h,nvars(r)-1);`

	l	l+1		h
n:	h^n(F(l))	h^n(F(l+1))	......	h^n(F(h))
	....			
1:	h^1(F(l))	h^1(F(l+1))	......	h^1(F(h))
0:	h^0(F(l))	h^0(F(l+1))	......	h^0(F(h))
chi:	chi(F(l))	chi(F(l+1))	......	chi(F(h))

A `'-'` in the diagram refers to a zero entry; a `'*'` refers to a negative entry (= dimension not yet determined). refers to a not computed dimension.
If `printlevel`>=1, step-by step timings will be printed. If `printlevel`>=2 we add progress debug messages if `printlevel`>=3, even all intermediate results...

Note: This procedure is based on the Bernstein-Gel'fand-Gel'fand correspondence and on Tate resolution (see [Eisenbud, Floystad, Schreyer: Sheaf cohomology and free resolutions over exterior algebras, Trans AMS 355 (2003)]).
`sheafCohBGG(M,l,h)` does not compute all values in the above table. To determine all

values of `h^i(F(d))`, `d=l..h`, use `sheafCohBGG(M,l-n,h+n)`. Experimental version. Should require less memory.

Example:

```
LIB "sheafcoh.lib";
int pl = printlevel;
int l,h, t;
//------------------------------------------
// cohomology of structure sheaf on P^4:
//------------------------------------------
ring r=32001,x(1..5),dp;
↦ // ** 32001 is invalid as characteristic of the ground field. 32003 is us\
   ed.
module M= getStructureSheaf(); // OO_P^4
l = -12; h = 12; // range of twists: l..h
printlevel = 0;
////////////////////////////////////////////
t = timer;
def A = sheafCoh(M, l, h); // global Ext method:
displayCohom(A,l,h,nvars(basering)-1);
↦         -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2   \
    -1     0     1     2     3     4     5     6     7     8     9    10    \
   11    12
↦ ---------------------------------------------------------------------------------
↦    4:   330   210   126    70    35    15     5     1     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦    3:     -     -     -     -     -     -     -     -     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦    2:     -     -     -     -     -     -     -     -     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦    1:     -     -     -     -     -     -     -     -     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦    0:     -     -     -     -     -     -     -     -     -     -     -   \
     -     1     5    15    35    70   126   210   330   495   715  1001  13\
   65  1820
↦ ---------------------------------------------------------------------------------
↦ chi:   330   210   126    70    35    15     5     1     0     0     0   \
     0     1     5    15    35    70   126   210   330   495   715  1001  13\
   65  1820
"Time: ", timer - t;
↦ Time:  5
////////////////////////////////////////////
t = timer;
A = sheafCohBGG(M, l, h);  // BGG method (without optimization):
displayCohom(A,l,h,nvars(basering)-1);
↦         -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2   \
    -1     0     1     2     3     4     5     6     7     8     9    10    \
   11    12
↦ ---------------------------------------------------------------------------------
↦    4:   330   210   126    70    35    15     5     1     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     *     *     *     *
↦    3:     *     -     -     -     -     -     -     -     -     -     -   \
     -     -     -     -     -     -     -     -     -     -     -     *     *     *
```

```
↦   2:     *     *     -     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     *     *
↦   1:     *     *     *     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     *
↦   0:     *     *     *     *     -     -     -     -     -     -     -   \
    -     1     5    15    35    70   126   210   330   495   715  1001  13\
  65  1820
↦ ---------------------------------------------------------------------------------
↦ chi:     *     *     *     *    35    15     5     1     0     0     0   \
    0     1     5    15    35    70   126   210   330   495     *     *   \
    *     *
"Time: ", timer - t;
↦ Time:  14
//////////////////////////////////////////////
t = timer;
A = sheafCohBGG2(M, l, h); // BGG method (with optimization)
↦ Cohomology table:
↦          -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2   \
   -1     0     1     2     3     4     5     6     7     8     9    10    \
  11    12
↦ ---------------------------------------------------------------------------------
↦   4:   330   210   126    70    35    15     5     1     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   3:     *     -     -     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   2:     *     *     -     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   1:     *     *     *     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   0:     *     *     *     *     -     -     -     -     -     -     -   \
    -     1     5    15    35    70   126   210   330   495   715  1001  13\
  65  1820
↦ ---------------------------------------------------------------------------------
↦ chi:     *     *     *     *    35    15     5     1     0     0     0   \
    0     1     5    15    35    70   126   210   330   495   715  1001  13\
  65  1820
displayCohom(A,l,h,nvars(basering)-1);
↦          -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2   \
   -1     0     1     2     3     4     5     6     7     8     9    10    \
  11    12
↦ ---------------------------------------------------------------------------------
↦   4:   330   210   126    70    35    15     5     1     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   3:     *     -     -     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   2:     *     *     -     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   1:     *     *     *     -     -     -     -     -     -     -     -   \
    -     -     -     -     -     -     -     -     -     -     -     -     -     -
↦   0:     *     *     *     *     -     -     -     -     -     -     -   \
    -     1     5    15    35    70   126   210   330   495   715  1001  13\
  65  1820
↦ ---------------------------------------------------------------------------------
↦ chi:     *     *     *     *    35    15     5     1     0     0     0    \
     0     1     5    15    35    70   126   210   330   495   715  1001  13\
  65  1820
"Time: ", timer - t;
↦ Time:  4
/////////////////////////////////////////////
printlevel = pl;
kill A, r;
//------------------------------------------
// cohomology of cotangential bundle on P^3:
//------------------------------------------
ring R=32001,(x,y,z,u),dp;
↦ // ** 32001 is invalid as characteristic of the ground field. 32003 is us\
  ed.
module M = getCotangentialBundle();
l = -12; h = 11; // range of twists: l..h
/////////////////////////////////////////////
printlevel = 0;
t = timer;
def B = sheafCoh(M, l, h); // global Ext method:
displayCohom(B,l,h,nvars(basering)-1);
↦       -12  -11  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1    0    \
  1    2    3    4    5    6    7    8    9   10   11
↦ ----------------------------------------------------------------------------------
↦   3:  715  540  396  280  189  120   70   36   15    4    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   2:    -    -    -    -    -    -    -    -    -    -    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   1:    -    -    -    -    -    -    -    -    -    -    -    -    1    \
  -    -    -    -    -    -    -    -    -    -    -
↦   0:    -    -    -    -    -    -    -    -    -    -    -    -    -    \
  -    6   20   45   84  140  216  315  440  594  780
↦ ----------------------------------------------------------------------------------
↦ chi: -715 -540 -396 -280 -189 -120  -70  -36  -15   -4    0    0   -1    \
  0    6   20   45   84  140  216  315  440  594  780
"Time: ", timer - t;
↦ Time:  2
/////////////////////////////////////////////
t = timer;
B = sheafCohBGG(M, l, h);  // BGG method (without optimization):
displayCohom(B,l,h,nvars(basering)-1);
↦       -12  -11  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1    0    \
  1    2    3    4    5    6    7    8    9   10   11
↦ ----------------------------------------------------------------------------------
↦   3:  715  540  396  280  189  120   70   36   15    4    -    -    -    \
  -    -    -    -    -    -    -    -    *    *    *
↦   2:    *    -    -    -    -    -    -    -    -    -    -    -    -    \
  -    -    -    -    -    -    -    -    -    *    *
↦   1:    *    *    -    -    -    -    -    -    -    -    -    -    1    \
  -    -    -    -    -    -    -    -    -    -    *
↦   0:    *    *    *    -    -    -    -    -    -    -    -    -    -    \
  -    6   20   45   84  140  216  315  440  594  780
↦ ----------------------------------------------------------------------------------
↦ chi:    *    *    * -280 -189 -120  -70  -36  -15   -4    0    0   -1    \
  0    6   20   45   84  140  216  315    *    *    *
"Time: ", timer - t;
↦ Time:  2
//////////////////////////////////////////////
t = timer;
B = sheafCohBGG2(M, l, h); // BGG method (with optimization)
↦ Cohomology table:
↦       -12  -11  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1    0    \
  1    2    3    4    5    6    7    8    9   10   11
↦ ---------------------------------------------------------------------------------
↦   3:  715  540  396  280  189  120   70   36   15    4    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   2:    *    -    -    -    -    -    -    -    -    -    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   1:    *    *    -    -    -    -    -    -    -    -    -    -    1    \
  -    -    -    -    -    -    -    -    -    -    -
↦   0:    *    *    *    -    -    -    -    -    -    -    -    -    -    \
  -    6   20   45   84  140  216  315  440  594  780
↦ ---------------------------------------------------------------------------------
↦ chi:    *    *    * -280 -189 -120  -70  -36  -15   -4    0    0   -1    \
  0    6   20   45   84  140  216  315  440  594  780
displayCohom(B,l,h,nvars(basering)-1);
↦       -12  -11  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1    0    \
  1    2    3    4    5    6    7    8    9   10   11
↦ ---------------------------------------------------------------------------------
↦   3:  715  540  396  280  189  120   70   36   15    4    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   2:    *    -    -    -    -    -    -    -    -    -    -    -    -    \
  -    -    -    -    -    -    -    -    -    -    -
↦   1:    *    *    -    -    -    -    -    -    -    -    -    -    1    \
  -    -    -    -    -    -    -    -    -    -    -
↦   0:    *    *    *    -    -    -    -    -    -    -    -    -    -    \
  -    6   20   45   84  140  216  315  440  594  780
↦ ---------------------------------------------------------------------------------
↦ chi:    *    *    * -280 -189 -120  -70  -36  -15   -4    0    0   -1    \
  0    6   20   45   84  140  216  315  440  594  780
"Time: ", timer - t;
↦ Time:  0
//////////////////////////////////////////////
printlevel = pl;
```

See also: Section D.5.18.12 [displayCohom], page 1619; Section D.5.18.4 [sheafCohBGG], page 1608.

D.5.18.8 sheafCoh

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCoh(M,l,h); M module, l,h int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute, `h>=l`. The basering `S` has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Display: The intmat can be displayed in a diagram of the following form: with `displayCohom(A,l,h,nvars(r)-1);`

```
                l          l+1                  h
      ------------------------------------------------
       n:   h^n(F(l))   h^n(F(l+1))   ......   h^n(F(h))
            ...........................................
       1:   h^1(F(l))   h^1(F(l+1))   ......   h^1(F(h))
       0:   h^0(F(l))   h^0(F(l+1))   ......   h^0(F(h))
      ------------------------------------------------
     chi:   chi(F(l))   chi(F(l+1))   ......   chi(F(h))
```

A '-' in the diagram refers to a zero entry.

Note: The procedure is based on local duality as described in [Eisenbud: Computing cohomology. In Vasconcelos: Computational methods in commutative algebra and algebraic geometry. Springer (1998)].
By default, the procedure uses `mres` to compute the Ext modules. If called with the additional parameter `"sres"`, the `sres` command is used instead.

Example:

```
LIB "sheafcoh.lib";
//
// cohomology of structure sheaf on P^4:
//-------------------------------------------
ring r=0,x(1..5),dp;
module M=0;
intmat A=sheafCoh(0,-7,2);
A;
↦ 15,5,1,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,1,5,15
displayCohom(A,-7,2,nvars(r)-1);
↦        -7  -6  -5  -4  -3  -2  -1   0   1   2
↦ ------------------------------------------------
↦    4:  15   5   1   -   -   -   -   -   -   -
↦    3:   -   -   -   -   -   -   -   -   -   -
↦    2:   -   -   -   -   -   -   -   -   -   -
↦    1:   -   -   -   -   -   -   -   -   -   -
↦    0:   -   -   -   -   -   -   -   1   5  15
↦ ------------------------------------------------
↦ chi:   15   5   1   0   0   0   0   1   5  15
//
// cohomology of cotangential bundle on P^3:
//-------------------------------------------
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
attrib(M,"isHomog",v);
intmat B=sheafCoh(M,-6,2);
```

```
displayCohom(B,-6,2,nvars(R)-1);
↦        -6   -5   -4   -3   -2   -1    0    1    2
↦ ---------------------------------------------------
↦    3:  70   36   15    4    -    -    -    -    -
↦    2:   -    -    -    -    -    -    -    -    -
↦    1:   -    -    -    -    -    -    1    -    -
↦    0:   -    -    -    -    -    -    -    -    6
↦ ---------------------------------------------------
↦ chi: -70  -36  -15   -4    0    0   -1    0    6
```

See also: Section D.5.18.10 [dimH], page 1618; Section D.5.18.4 [sheafCohBGG], page 1608; Section D.5.18.9 [sheafCoh_w], page 1617.

D.5.18.9 sheafCoh_w

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: sheafCoh(M,l,h); M module, l,h int, w intvec

Assume: `M` is graded by w, `h>=l`. The basering `S` has `n+1` variables.

Return: intmat, cohomology of twists of the coherent sheaf F on P^n associated to coker(M). The range of twists is determined by `l`, `h`.

Display: The intmat can be displayed in a diagram of the following form: with `displayCohom(A,l,h,nvars(r)-1);`

```
                 l           l+1                     h
      ----------------------------------------------------
      n:    h^n(F(l))    h^n(F(l+1))    ......    h^n(F(h))
              ..............................................
      1:    h^1(F(l))    h^1(F(l+1))    ......    h^1(F(h))
      0:    h^0(F(l))    h^0(F(l+1))    ......    h^0(F(h))
      ----------------------------------------------------
    chi:    chi(F(l))    chi(F(l+1))    ......    chi(F(h))
```

A `'-'` in the diagram refers to a zero entry.

Note: The procedure is based on local duality as described in [Eisenbud: Computing cohomology. In Vasconcelos: Computational methods in commutative algebra and algebraic geometry. Springer (1998)].
By default, the procedure uses `mres` to compute the Ext modules. If called with the additional parameter `"sres"`, the `sres` command is used instead.

Example:

```
LIB "sheafcoh.lib";
//
// cohomology of structure sheaf on P^4:
//-------------------------------------------
ring r=0,x(1..5),dp;
module M=0;
intmat A=sheafCoh_w(0,-7,2,intvec(0));
A;
↦ 15,5,1,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,1,5,15
displayCohom(A,-7,2,nvars(r)-1);
↦       -7  -6  -5  -4  -3  -2  -1   0   1   2
↦ --------------------------------------------
↦   4:  15   5   1   -   -   -   -   -   -   -
↦   3:   -   -   -   -   -   -   -   -   -   -
↦   2:   -   -   -   -   -   -   -   -   -   -
↦   1:   -   -   -   -   -   -   -   -   -   -
↦   0:   -   -   -   -   -   -   -   1   5  15
↦ --------------------------------------------
↦ chi:  15   5   1   0   0   0   0   1   5  15
//
// cohomology of cotangential bundle on P^3:
//-------------------------------------------
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
intmat B=sheafCoh_w(M,-6,2,v);
displayCohom(B,-6,2,nvars(R)-1);
↦       -6  -5  -4  -3  -2  -1   0   1   2
↦ ----------------------------------------
↦   3:  70  36  15   4   -   -   -   -   -
↦   2:   -   -   -   -   -   -   -   -   -
↦   1:   -   -   -   -   -   -   1   -   -
↦   0:   -   -   -   -   -   -   -   -   6
↦ ----------------------------------------
↦ chi: -70 -36 -15  -4   0   0  -1   0   6
```

See also: Section D.5.18.10 [dimH], page 1618; Section D.5.18.8 [sheafCoh], page 1615; Section D.5.18.4 [sheafCohBGG], page 1608.

D.5.18.10 dimH

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: dimH(i,M,d); M module, i,d int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute, `h>=1`, and the basering `S` has `n+1` variables.

Return: int, vector space dimension of $H^i(F(d))$ for F the coherent sheaf on P^n associated to coker(M).

Note: The procedure is based on local duality as described in [Eisenbud: Computing cohomology. In Vasconcelos: Computational methods in commutative algebra and algebraic geometry. Springer (1998)].

Example:

```
LIB "sheafcoh.lib";
ring R=0,(x,y,z,u),dp;
resolution T1=mres(maxideal(1),0);
module M=T1[3];
intvec v=2,2,2,2,2,2;
attrib(M,"isHomog",v);
```

```
dimH(0,M,2);
↦ 6
dimH(1,M,0);
↦ 1
dimH(2,M,1);
↦ 0
dimH(3,M,-5);
↦ 36
```

See also: Section D.5.18.8 [sheafCoh], page 1615; Section D.5.18.4 [sheafCohBGG], page 1608.

D.5.18.11 dimGradedPart

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: dimGradedPart(M,d); M module, d int

Assume: `M` is graded, and it comes assigned with an admissible degree vector as an attribute

Return: int

Note: Output is the vector space dimension of the graded part of degree d of coker(M).

Example:

```
LIB "sheafcoh.lib";
ring R=0,(x,y,z),dp;
module M=maxideal(3);
// assign compatible weight vector (here: 0)
homog(M);
↦ 1
// compute dimension of graded pieces of R/<x,y,z>^3 :
dimGradedPart(M,0);
↦ 1
dimGradedPart(M,1);
↦ 3
dimGradedPart(M,2);
↦ 6
dimGradedPart(M,3);
↦ 0
// shift grading:
attrib(M,"isHomog",intvec(2));
dimGradedPart(M,2);
↦ 1
```

D.5.18.12 displayCohom

Procedure from library `sheafcoh.lib` (see Section D.5.18 [sheafcoh_lib], page 1604).

Usage: displayCohom(data,l,h,n); data intmat, l,h,n int

Assume: `h>=l`, `data` is the return value of `sheafCoh(M,l,h)` or of `sheafCohBGG(M,l,h)`, and the basering has `n+1` variables.

Return: none

Note: The intmat is displayed in a diagram of the following form:

```
              l          l+1                 h
   ------------------------------------------------
   n:     h^n(F(l))   h^n(F(l+1))   ......   h^n(F(h))
          ...........................................
   1:     h^1(F(l))   h^1(F(l+1))   ......   h^1(F(h))
   0:     h^0(F(l))   h^0(F(l+1))   ......   h^0(F(h))
   ------------------------------------------------
   chi:   chi(F(l))   chi(F(l+1))   ......   chi(F(h))
```

where `F` refers to the associated sheaf of `M` on P^n.
A `'-'` in the diagram refers to a zero entry, a `'*'` refers to a negative entry (= dimension not yet determined).

D.5.19 JMBTest_lib

Library: JMBTest.lib

Purpose: A library for Singular which performs JM basis test.

Author: Michela Ceria, email: michela.ceria@unito.it

Overview: The library performs the J-marked basis test, as described in [CR], [BCLR]. Such a test is performed via the criterion explained in [BCLR], concerning Eliahou-Kervaire polynomials (EK from now on). We point out that all the polynomials are homogeneous and they must be arranged by degree.
The fundamental steps are the following:
-construct the Vm polynomials, via the algorithm VConstructor explained in [CR];
-construct the Eliahou-Kervaire polynomials defined in [BCLR];
-reduce the Eliahou-Kervaire polynomials using the Vm's;
-if it exist an Eliahou-Kervaire polynomial such that its reduction mod Vm is different from zero, the given one is not a J-Marked basis.

The algorithm terminates only if the ordering is ip. Anyway, the number of reduction steps is bounded.

References:
[CR] Francesca Cioffi, Margherita Roggero,Flat Families by Strongly Stable Ideals and a Generalization of Groebner Bases, J. Symbolic Comput. 46, 1070-1084, (2011).
[BCLR] Cristina Bertone, Francesca Cioffi, Paolo Lella, Margherita Roggero, Upgraded methods for the effective computation of marked schemes on a strongly stable ideal, Journal of Symbolic Computation
(2012), http://dx.doi.org/10.1016/j.jsc.2012.07.006

Procedures: See also: Section D.5.20 [JMSConst_lib], page 1622.

D.5.19.1 Minimus

Procedure from library `JMBTest.lib` (see Section D.5.19 [JMBTest_lib], page 1620).

Usage: Minimus(L); G list, c int

Return: list: V

Notes: it returns the minimal variable generating the ideal L.
The input must be an ideal generated by variables.

Example:

```
LIB "JMBTest.lib";
ring r=0, (x,y,z), ip;
ideal I=y,x,z;
Minimus(I);
↦ x
```

D.5.19.2 Maximus

Procedure from library `JMBTest.lib` (see Section D.5.19 [JMBTest_lib], page 1620).

Usage: Maximus(L); G list, c int

Return: list: V

Notes: it returns the maximal variable generating the ideal L.
The input must be an ideal generated by variables.

Example:

```
LIB "JMBTest.lib";
ring r=0, (x,y,z), ip;
ideal I=y,x,z;
Maximus(I);
↦ z
```

D.5.19.3 StartOrderingV

Procedure from library `JMBTest.lib` (see Section D.5.19 [JMBTest_lib], page 1620).

Usage: StartOrdina(V,G); V list, G list

Return: list: R

Note: Input Vm,G. This procedure uses OrderingV to get
the ordered polynomials as in [BCLR].

Example:

```
LIB "JMBTest.lib";
ring r=0, (x,y,z), ip;
jmp r1;
r1.h=z^3;
r1.t=poly(0);
jmp r2;
r2.h=z^2*y;
r2.t=poly(0);
jmp r3;
r3.h=z*y^2;
r3.t=-x^2*y;
jmp r4;
r4.h=y^5;
r4.t=poly(0);
list G2F=list(list(r1,r2,r3),list(r4));
StartOrderingV(VConst(G2F,4,basering)[1],G2F);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       3
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       2
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.5.19.4 TestJMark

Procedure from library `JMBTest.lib` (see Section D.5.19 [JMBTest_lib], page 1620).

Usage: TestJMark(G); G list

Return: int: i

Note: This procedure performs J-marked basis test.
The input is a list of J-marked polynomials (jmp) arranged by degree, so G1 is a list of list.
The output is a boolean evaluation:
True=1/False=0

Example:

```
LIB "JMBTest.lib";
ring r=0, (x,y,z), ip;
jmp r1;
r1.h=z^3;
r1.t=poly(0);
jmp r2;
r2.h=z^2*y;
r2.t=poly(0);
jmp r3;
r3.h=z*y^2 ;
r3.t=-x^2*y;
jmp r4;
r4.h=y^5;
r4.t=poly(0);
list G2F=list(list(r1,r2,r3),list(r4));
TestJMark(G2F,r);
↦ NOT A BASIS
↦ 0
```

D.5.20 JMSConst_lib

Library: JMSConst.lib

Purpose: A library for Singular which constructs J-Marked Schemes.

Author: Michela Ceria, email: michela.ceria@unito.it

Overview: The library performs the J-marked computation, as described in [BCLR]. As in JMBTest.lib we construct the V polynomials and we reduce the EK polynomials w.r.t. them, putting the coefficients as results.

The algorithm terminates only if the ordering is ip. Anyway, the number of reduction steps is bounded.

References:

[CR] Francesca Cioffi, Margherita Roggero,Flat Families by Strongly Stable Ideals and a Generalization of Groebner Bases, J. Symbolic Comput. 46, 1070-1084, (2011).
[BCLR] Cristina Bertone, Francesca Cioffi, Paolo Lella, Margherita Roggero, Upgraded methods for the effective computation of marked schemes on a strongly stable ideal, Journal of Symbolic Computation
(2012), http://dx.doi.org/10.1016/j.jsc.2012.07.006

Procedures: See also: Section D.5.19 [JMBTest_lib], page 1620; Section D.5.20 [JMSConst_lib], page 1622.

D.5.20.1 BorelCheck

Procedure from library `JMSConst.lib` (see Section D.5.20 [JMSConst_lib], page 1622).

Usage: BorelCheck(Borid,r); Borid ideal, r ring

Return: int: d

Note: Input must be a monomial ideal.
The procedure checks whether the Borel moves produce elements belonging to Borid.

Example:

```
LIB "JMSConst.lib";
ring r=0, (x,y,z),ip;
ideal Borid=y^2*z,y*z^2,z^3,y^5;
BorelCheck(Borid,r);
↦ 1
```

D.5.20.2 JMarkedScheme

Procedure from library `JMSConst.lib` (see Section D.5.20 [JMSConst_lib], page 1622).

Usage: JMarkedScheme(Borid, r); Borid ideal, r ring

Return: list: Jms

Note: This procedure performs automatically the whole construction of the J-marked scheme.

Example:

```
LIB "JMSConst.lib";
ring r=0, (x,y,z),ip;
ideal Borid=y^2*z,y*z^2,z^3,y^5;
JMarkedScheme(Borid,r);
↦ [1]:
```

```
↦    (-c(1)*c(15)+c(1)*c(9)+c(8)^2+c(1)*c(4)*c(8)-c(2)*c(8)-c(1)*c(5)+c(1)*\
   c(2)*c(4)+c(3))
↦ [2]:
↦    (-c(1)*c(16)-c(10)+c(8)*c(9)+c(1)*c(4)*c(9)+c(6)-c(2)*c(5)-c(3)*c(4)+c\
   (2)^2*c(4))
↦ [3]:
↦    (-c(1)*c(17)+c(8)*c(10)+c(1)*c(4)*c(10)-c(2)*c(10)+c(3)*c(9)+c(7)-c(3)\
   *c(5)+c(2)*c(3)*c(4))
↦ [4]:
↦    (-c(11)-c(4)^2)
↦ [5]:
↦    (-c(1)*c(18)-c(12)+c(8)*c(11)+c(1)*c(4)*c(11)-c(2)*c(11)+c(4)*c(9)-2*c\
   (4)*c(5)+c(2)*c(4)^2)
↦ [6]:
↦    (-c(1)*c(19)-c(13)+c(8)*c(12)+c(1)*c(4)*c(12)-c(2)*c(12)+c(5)*c(9)-c(4\
   )*c(6)-c(5)^2+c(2)*c(4)*c(5))
↦ [7]:
↦    (-c(1)*c(20)-c(14)+c(8)*c(13)+c(1)*c(4)*c(13)-c(2)*c(13)+c(6)*c(9)-c(4\
   )*c(7)-c(5)*c(6)+c(2)*c(4)*c(6))
↦ [8]:
↦    (-c(1)*c(21)+c(8)*c(14)+c(1)*c(4)*c(14)-c(2)*c(14)+c(7)*c(9)-c(5)*c(7)\
   +c(2)*c(4)*c(7))
↦ [9]:
↦    (c(1)*c(16)-c(1)*c(12)+c(1)*c(8)*c(11)+c(1)*c(2)*c(11)+c(10)-c(8)*c(9)\
   )
↦ [10]:
↦    (-c(17)-c(8)*c(16)+c(2)*c(16)+c(9)*c(15)+c(13)-c(2)*c(12)+c(1)*c(9)*c(\
   11)-c(3)*c(11)+c(2)^2*c(11)-c(9)^2)
↦ [11]:
↦    (-c(8)*c(17)+c(3)*c(16)+c(10)*c(15)+c(14)-c(3)*c(12)+c(1)*c(10)*c(11)+\
   c(2)*c(3)*c(11)-c(9)*c(10))
↦ [12]:
↦    (-c(18)-c(4)*c(11))
↦ [13]:
↦    (-c(19)-c(8)*c(18)+c(4)*c(16)+c(11)*c(15)-c(4)*c(12)+c(1)*c(11)^2-c(9)\
   *c(11)-c(5)*c(11)+c(2)*c(4)*c(11))
↦ [14]:
↦    (-c(20)-c(8)*c(19)+c(5)*c(16)+c(12)*c(15)+c(1)*c(11)*c(12)-c(9)*c(12)-\
   c(5)*c(12)-c(6)*c(11)+c(2)*c(5)*c(11))
↦ [15]:
↦    (-c(21)-c(8)*c(20)+c(6)*c(16)+c(13)*c(15)+c(1)*c(11)*c(13)-c(9)*c(13)-\
   c(6)*c(12)-c(7)*c(11)+c(2)*c(6)*c(11))
↦ [16]:
↦    (-c(8)*c(21)+c(7)*c(16)+c(14)*c(15)+c(1)*c(11)*c(14)-c(9)*c(14)-c(7)*c\
   (12)+c(2)*c(7)*c(11))
↦ [17]:
↦    (-c(1)*c(27)+c(1)*c(8)*c(26)+c(1)*c(2)*c(26)-c(1)^2*c(9)*c(25)-c(1)*c(\
   8)^2*c(25)-c(1)*c(2)*c(8)*c(25)+c(1)*c(3)*c(25)-c(1)*c(2)^2*c(25)+c(24)-c\
   (8)*c(23)-c(1)*c(4)*c(23)-c(15)*c(22)-c(1)*c(11)*c(22)-c(4)*c(8)*c(22)+c(\
   5)*c(22)-c(2)*c(4)*c(22)-c(1)^2*c(10)+2*c(1)^2*c(8)*c(9)+2*c(1)^2*c(2)*c(\
   9)+c(1)*c(8)^3+c(1)*c(2)*c(8)^2-c(1)*c(3)*c(8)+c(1)*c(2)^2*c(8)-2*c(1)*c(\
   2)*c(3)+c(1)*c(2)^3)
↦ [18]:
```

```
↦    (c(28)-c(2)*c(27)+c(1)*c(9)*c(26)-c(3)*c(26)+c(2)^2*c(26)+c(1)*c(10)*c\
   (25)-c(1)*c(8)*c(9)*c(25)-2*c(1)*c(2)*c(9)*c(25)+2*c(2)*c(3)*c(25)-c(2)^3\
   *c(25)+c(4)*c(24)-c(1)*c(11)*c(23)-c(9)*c(23)+c(5)*c(23)-2*c(2)*c(4)*c(23\
   )-c(16)*c(22)-c(4)*c(9)*c(22)-c(1)*c(8)*c(10)-2*c(1)*c(2)*c(10)+c(1)^2*c(\
   9)^2+c(1)*c(8)^2*c(9)+2*c(1)*c(2)*c(8)*c(9)-2*c(1)*c(3)*c(9)+3*c(1)*c(2)^\
   2*c(9)+c(3)^2-3*c(2)^2*c(3)+c(2)^4)
↦ [19]:
↦    (c(29)-c(3)*c(27)+c(1)*c(10)*c(26)+c(2)*c(3)*c(26)-c(1)*c(8)*c(10)*c(2\
   5)-c(1)*c(2)*c(10)*c(25)-c(1)*c(3)*c(9)*c(25)+c(3)^2*c(25)-c(2)^2*c(3)*c(\
   25)-c(1)*c(11)*c(24)+c(5)*c(24)-c(2)*c(4)*c(24)-c(10)*c(23)-c(3)*c(4)*c(2\
   3)-c(17)*c(22)-c(4)*c(10)*c(22)+c(1)^2*c(9)*c(10)+c(1)*c(8)^2*c(10)+c(1)*\
   c(2)*c(8)*c(10)-2*c(1)*c(3)*c(10)+c(1)*c(2)^2*c(10)+c(1)*c(3)*c(8)*c(9)+2\
   *c(1)*c(2)*c(3)*c(9)-2*c(2)*c(3)^2+c(2)^3*c(3))
↦ [20]:
↦    (c(1)*c(12)-c(1)*c(8)*c(11)-c(1)*c(2)*c(11)-c(1)*c(4)*c(9)-c(6)+c(2)*c\
   (5)+c(3)*c(4)-c(2)^2*c(4))
↦ [21]:
↦    (c(1)*c(12)*c(25)-c(1)*c(8)*c(11)*c(25)-c(1)*c(2)*c(11)*c(25)-c(1)*c(4\
   )*c(9)*c(25)-c(6)*c(25)+c(2)*c(5)*c(25)+c(3)*c(4)*c(25)-c(2)^2*c(4)*c(25)\
   -c(11)*c(23)-c(4)^2*c(23)-c(18)*c(22)-c(4)*c(11)*c(22)+c(1)*c(13)-c(1)*c(\
   8)*c(12)-c(1)*c(2)*c(12)+c(1)^2*c(9)*c(11)+c(1)*c(8)^2*c(11)+c(1)*c(2)*c(\
   8)*c(11)-c(1)*c(3)*c(11)+c(1)*c(2)^2*c(11)-c(1)*c(4)*c(10)+c(1)*c(4)*c(8)\
   *c(9)-c(1)*c(5)*c(9)+2*c(1)*c(2)*c(4)*c(9)-c(7)+c(2)*c(6)+c(3)*c(5)-c(2)^\
   2*c(5)-2*c(2)*c(3)*c(4)+c(2)^3*c(4))
↦ [22]:
↦    (c(4)*c(28)-c(1)*c(11)*c(27)-c(2)*c(4)*c(27)+c(1)*c(12)*c(26)-c(6)*c(2\
   6)+c(2)*c(5)*c(26)+c(1)*c(13)*c(25)-c(1)*c(8)*c(12)*c(25)-c(1)*c(2)*c(12)\
   *c(25)-c(1)*c(5)*c(9)*c(25)-c(7)*c(25)+c(2)*c(6)*c(25)+c(3)*c(5)*c(25)-c(\
   2)^2*c(5)*c(25)-c(12)*c(23)-c(4)*c(5)*c(23)-c(19)*c(22)-c(4)*c(12)*c(22)+\
   c(1)*c(14)-c(1)*c(8)*c(13)-c(1)*c(2)*c(13)+c(1)^2*c(9)*c(12)+c(1)*c(8)^2*\
   c(12)+c(1)*c(2)*c(8)*c(12)-c(1)*c(3)*c(12)+c(1)*c(2)^2*c(12)-c(1)*c(5)*c(\
   10)+c(1)*c(5)*c(8)*c(9)-c(1)*c(6)*c(9)+2*c(1)*c(2)*c(5)*c(9)+c(2)*c(7)+c(\
   3)*c(6)-c(2)^2*c(6)-2*c(2)*c(3)*c(5)+c(2)^3*c(5))
↦ [23]:
↦    (c(4)*c(29)-c(1)*c(11)*c(28)+c(5)*c(28)-c(2)*c(4)*c(28)-c(6)*c(27)+c(1\
   )*c(13)*c(26)-c(7)*c(26)+c(2)*c(6)*c(26)+c(1)*c(14)*c(25)-c(1)*c(8)*c(13)\
   *c(25)-c(1)*c(2)*c(13)*c(25)-c(1)*c(6)*c(9)*c(25)+c(2)*c(7)*c(25)+c(3)*c(\
   6)*c(25)-c(2)^2*c(6)*c(25)-c(13)*c(23)-c(4)*c(6)*c(23)-c(20)*c(22)-c(4)*c\
   (13)*c(22)-c(1)*c(8)*c(14)-c(1)*c(2)*c(14)+c(1)^2*c(9)*c(13)+c(1)*c(8)^2*\
   c(13)+c(1)*c(2)*c(8)*c(13)-c(1)*c(3)*c(13)+c(1)*c(2)^2*c(13)-c(1)*c(6)*c(\
   10)+c(1)*c(6)*c(8)*c(9)-c(1)*c(7)*c(9)+2*c(1)*c(2)*c(6)*c(9)+c(3)*c(7)-c(\
   2)^2*c(7)-2*c(2)*c(3)*c(6)+c(2)^3*c(6))
↦ [24]:
↦    (-c(1)*c(11)*c(29)+c(5)*c(29)-c(2)*c(4)*c(29)-c(7)*c(27)+c(1)*c(14)*c(\
   26)+c(2)*c(7)*c(26)-c(1)*c(8)*c(14)*c(25)-c(1)*c(2)*c(14)*c(25)-c(1)*c(7)\
   *c(9)*c(25)+c(3)*c(7)*c(25)-c(2)^2*c(7)*c(25)-c(14)*c(23)-c(4)*c(7)*c(23)\
   -c(21)*c(22)-c(4)*c(14)*c(22)+c(1)^2*c(9)*c(14)+c(1)*c(8)^2*c(14)+c(1)*c(\
   2)*c(8)*c(14)-c(1)*c(3)*c(14)+c(1)*c(2)^2*c(14)-c(1)*c(7)*c(10)+c(1)*c(7)\
   *c(8)*c(9)+2*c(1)*c(2)*c(7)*c(9)-2*c(2)*c(3)*c(7)+c(2)^3*c(7))
```

D.6 Singularities

D.6.1 alexpoly_lib

Library: alexpoly.lib

Purpose: Resolution Graph and Alexander Polynomial

Author: Fernando Hernando Carrillo, hernando@agt.uva.es
Thomas Keilen, keilen@mathematik.uni-kl.de

Overview: A library for computing the resolution graph of a plane curve singularity f, the total multiplicities of the total transforms of the branches of f along the exceptional divisors of a minimal good resolution of f, the Alexander polynomial of f, and the zeta function of its monodromy operator.

Procedures:

D.6.1.1 resolutiongraph

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: resolutiongraph(INPUT); INPUT poly or list

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by `hnexpansion(f[,"ess"])`, or the output of `develop(f)` resp. of `extdevelop(f,n)`, or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity.

Return: intmat, the incidence matrix of the resolution graph of the plane curve defined by INPUT, where the entries on the diagonal are the weights of the vertices of the graph and a negative entry corresponds to the strict transform of a branch of the curve.

Note: In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.
If you are not sure whether the INPUT polynomial is reduced or not, use `squarefree(INPUT)` as input instead.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
poly f1=(y2-x3)^2-4x5y-x7;
poly f2=y2-x3;
poly f3=y3-x2;
resolutiongraph(f1*f2*f3);
↦ 1,0,1,0,0,0,0,0,1,0,
↦ 0,2,1,0,0,0,0,0,0,0,
↦ 1,1,3,0,1,0,0,0,0,0,
↦ 0,0,0,4,1,0,1,0,0,0,
↦ 0,0,1,1,5,1,0,0,0,0,
↦ 0,0,0,0,1,-1,0,0,0,0,
↦ 0,0,0,1,0,0,-2,0,0,0,
↦ 0,0,0,0,0,0,0,2,1,0,
↦ 1,0,0,0,0,0,0,1,3,1,
↦ 0,0,0,0,0,0,0,0,1,-3
```

See also: Section D.6.1.3 [alexanderpolynomial], page 1628; Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.2 totalmultiplicities

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: totalmultiplicities(INPUT); INPUT poly or list

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by `hnexpansion(f[,"ess"])`, or the output of `develop(f)` resp. of `extdevelop(f,n)`, or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity.

Return: list `L` of three integer matrices. `L[1]` is the incidence matrix of the resolution graph of the plane curve defined by INPUT, where the entries on the diagonal are the weights of the vertices of the graph and a negative entry corresponds to the strict transform of a branch of the curve. `L[2]` is an integer matrix, which has for each vertex in the graph a row and for each branch of the curve a column. The entry in position [i,j] contains the total multiplicity of the j-th branch (i.e. the branch with weight -j in `L[1]`) along the exceptional divisor corresponding to the i-th row in `L[1]`. In particular, the i-th row contains the total multiplicities of the branches of the plane curve (defined by INPUT) along the exceptional divisor which corresponds to the i-th row in the incidence matrix `L[1]`. `L[3]` is an integer matrix which contains the (strict) multiplicities of the branches of the curve along the exceptional divisors in the same way as `L[2]` contains the total multiplicities.

Note: The total multiplicity of a branch along an exceptional divisor is the multiplicity with which this exceptional divisor occurs in the total transform of this branch under the resolution corresponding to the resolution graph.
In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.
If you are not sure whether the INPUT polynomial is reduced or not, use `squarefree(INPUT)` as input instead.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
poly f1=(y2-x3)^2-4x5y-x7;
poly f2=y2-x3;
poly f3=y3-x2;
totalmultiplicities(f1*f2*f3);
↦ [1]:
↦    1,0,1,0,0,0,0,0,1,0,
↦    0,2,1,0,0,0,0,0,0,0,
↦    1,1,3,0,1,0,0,0,0,0,
↦    0,0,0,4,1,0,1,0,0,0,
↦    0,0,1,1,5,1,0,0,0,0,
↦    0,0,0,0,1,-1,0,0,0,0,
↦    0,0,0,1,0,0,-2,0,0,0,
↦    0,0,0,0,0,0,0,2,1,0,
↦    1,0,0,0,0,0,0,1,3,1,
↦    0,0,0,0,0,0,0,0,1,-3
↦ [2]:
```

```
↦     4,2,2,
↦     6,3,2,
↦     12,6,4,
↦     13,7,4,
↦     26,13,8,
↦     0,0,0,
↦     0,0,0,
↦     4,2,3,
↦     8,4,6,
↦     0,0,0
↦  [3]:
↦     4,2,2,
↦     2,1,0,
↦     2,1,0,
↦     1,1,0,
↦     1,0,0,
↦     0,0,0,
↦     0,0,0,
↦     0,0,1,
↦     0,0,1,
↦     0,0,0
```

See also: Section D.6.1.3 [alexanderpolynomial], page 1628; Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.1.1 [resolutiongraph], page 1626.

D.6.1.3 alexanderpolynomial

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: alexanderpolynomial(INPUT); INPUT poly or list

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by `hnexpansion(f[,"ess"])`, or the output of `develop(f)` resp. of `extdevelop(f,n)`, or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity.

Create: a ring with variables t, t(1), ..., t(r) (where r is the number of branches of the plane curve singularity f defined by INPUT) and ordering ls over the ground field of the basering.
Moreover, the ring contains the Alexander polynomial of f in variables t(1), ..., t(r) (`alexpoly`), the zeta function of the monodromy operator of f in the variable t (`zeta_monodromy`), and a list containing the factors of the Alexander polynomial with multiplicities (`alexfactors`).

Return: a list, say `ALEX`, where `ALEX[1]` is the created ring

Note: to use the ring type: `def ALEXring=ALEX[i]; setring ALEXring;`.
Alternatively you may use the procedure sethnering and type: sethnering(ALEX,"ALEXring");
To access the Alexander polynomial resp. the zeta function resp. the factors of the Alexander polynomial type: `alexpoly` resp. `zeta_monodromy` resp. `alexfactors`.
In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as

input instead of the polynomial itself.
If you are not sure whether the INPUT polynomial is reduced or not, use squarefree(INPUT) as input instead.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
poly f1=(y2-x3)^2-4x5y-x7;
poly f2=y2-x3;
poly f3=y3-x2;
list ALEX=alexanderpolynomial(f1*f2*f3);
def ALEXring=ALEX[1];
setring ALEXring;
alexfactors;
↦ [1]:
↦    [1]:
↦       -t(1)^6*t(2)^3*t(3)^2+1
↦    [2]:
↦       -1
↦ [2]:
↦    [1]:
↦       -t(1)^12*t(2)^6*t(3)^4+1
↦    [2]:
↦       1
↦ [3]:
↦    [1]:
↦       -t(1)^26*t(2)^13*t(3)^8+1
↦    [2]:
↦       1
↦ [4]:
↦    [1]:
↦       -t(1)^4*t(2)^2*t(3)^3+1
↦    [2]:
↦       -1
↦ [5]:
↦    [1]:
↦       -t(1)^8*t(2)^4*t(3)^6+1
↦    [2]:
↦       1
alexpoly;
↦ -t(1)^36*t(2)^18*t(3)^13-t(1)^32*t(2)^16*t(3)^10-t(1)^30*t(2)^15*t(3)^11-\
   t(1)^26*t(2)^13*t(3)^8+t(1)^10*t(2)^5*t(3)^5+t(1)^6*t(2)^3*t(3)^2+t(1)^4*\
   t(2)^2*t(3)^3+1
zeta_monodromy;
↦ -t^67-t^58-t^56-t^47+t^20+t^11+t^9+1
```

See also: Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.4 semigroup

Procedure from library alexpoly.lib (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: semigroup(INPUT); INPUT poly or list

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by `hnexpansion(f[,"ess"])`, or the output of `develop(f)` resp. of `extdevelop(f,n)`, or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity.

Return: a list with three entries. The first and the second are lists `v_1,...,v_s` and `w_1,...,w_r` respectively of integer vectors such that the semigroup of the plane curve defined by the INPUT is generated by the vectors `v_1,...,v_s,w_1+k*e_1,...,w_r+k*e_r`, where e_i denotes the i-th standard basis vector and k runs through all non-negative integers. The third entry is the conductor of the plane curve singularity. Note that r is the number of branches of the plane curve singularity and integer vectors thus have size r.

Note: If the output is zero this means that the curve has one branch and is regular. In the reducible case the set of generators may not be minimal.
If you are not sure whether the INPUT polynomial is reduced or not, use `squarefree(INPUT)` as input instead.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
// Irreducible Case
semigroup((x2-y3)^2-4x5y-x7);
↦ [1]:
↦    [1]:
↦       4
↦    [2]:
↦       6
↦    [3]:
↦       17
↦ [2]:
↦    empty list
↦ [3]:
↦    20
// In the irreducible case, invariants() also calculates a minimal set of
// generators of the semigroup.
invariants((x2-y3)^2-4x5y-x7)[1][2];
↦ 4,6,17
// Reducible Case
poly f=(y2-x3)*(y2+x3)*(y4-2x3y2-4x5y+x6-x7);
semigroup(f);
↦ [1]:
↦    [1]:
↦       2,2,4
↦    [2]:
↦       3,3,6
↦ [2]:
↦    [1]:
↦       6,6,12
↦    [2]:
↦       6,7,13
↦    [3]:
```

```
↦        12,13,26
↦ [3]:
↦    20,21,41
```

See also: Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.5 proximitymatrix

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: proximitymatrix(INPUT); INPUT poly or list or intmat

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by `hnexpansion(f[,"ess"])`, or the output of `develop(f)` resp. of `extdevelop(f,n)`, or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity, or the resolution graph of a plane curve singularity (i.e. the output of resolutiongraph or the first entry in the output of totalmultiplicities).

Return: list, of three integer matrices. The first one is the proximity matrix of the plane curve defined by the INPUT, i.e. the entry i,j is 1 if the infinitely near point corresponding to row i is proximate to the infinitely near point corresponding to row j. The second integer matrix is the incidence matrix of the resolution graph of the plane curve. The entry on the diagonal in row i is -s-1 if s is the number of points proximate to the infinitely near point corresponding to the ith row in the matrix. The third integer matrix is the incidence matrix of the Enriques diagram of the plane curve singularity, i.e. each row corresponds to an infinitely near point in the minimal standard resolution, including the strict transforms of the branches, the diagonal element gives the level of the point, and the entry i,j is -1 if row i is proximate to row j.

Note: In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.
If you are not sure whether the INPUT polynomial is reduced or not, use `squarefree(INPUT)` as input instead.
If the input is a smooth curve, then the output will consist of three one-by-one zero matrices.
For the definitions of the computed objects see e.g. the book Eduardo Casas-Alvero, Singularities of Plane Curves.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
poly f1=(y2-x3)^2-4x5y-x7;
poly f2=y2-x3;
poly f3=y3-x2;
list proximity=proximitymatrix(f1*f2*f3);
/// The proximity matrix P ///
print(proximity[1]);
↦      1     0     0     0     0     0     0     0     0     0
↦     -1     1     0     0     0     0     0     0     0     0
↦     -1    -1     1     0     0     0     0     0     0     0
↦      0     0    -1     1     0     0     0     0     0     0
↦      0     0    -1    -1     1     0     0     0     0     0
↦      0     0     0     0    -1     1     0     0     0     0
↦      0     0     0    -1     0     0     1     0     0     0
↦     -1     0     0     0     0     0     0     1     0     0
↦     -1     0     0     0     0     0     0    -1     1     0
↦      0     0     0     0     0     0     0     0    -1     1
/// The proximity resolution graph N ///
print(proximity[2]);
↦     -5     0     1     0     0     0     0     0     1     0
↦      0    -2     1     0     0     0     0     0     0     0
↦      1     1    -3     0     1     0     0     0     0     0
↦      0     0     0    -3     1     0     1     0     0     0
↦      0     0     1     1    -2     1     0     0     0     0
↦      0     0     0     0     1    -1     0     0     0     0
↦      0     0     0     1     0     0    -1     0     0     0
↦      0     0     0     0     0     0     0    -2     1     0
↦      1     0     0     0     0     0     0     1    -2     1
↦      0     0     0     0     0     0     0     0     1    -1
/// They satisfy N=-transpose(P)*P ///
print(-transpose(proximity[1])*proximity[1]);
↦     -5     0     1     0     0     0     0     0     1     0
↦      0    -2     1     0     0     0     0     0     0     0
↦      1     1    -3     0     1     0     0     0     0     0
↦      0     0     0    -3     1     0     1     0     0     0
↦      0     0     1     1    -2     1     0     0     0     0
↦      0     0     0     0     1    -1     0     0     0     0
↦      0     0     0     1     0     0    -1     0     0     0
↦      0     0     0     0     0     0     0    -2     1     0
↦      1     0     0     0     0     0     0     1    -2     1
↦      0     0     0     0     0     0     0     0     1    -1
/// The incidence matrix of the Enriques diagram ///
print(proximity[3]);
↦      0     0     0     0     0     0     0     0     0     0
↦     -1     1     0     0     0     0     0     0     0     0
↦     -1    -1     2     0     0     0     0     0     0     0
↦      0     0    -1     3     0     0     0     0     0     0
↦      0     0    -1    -1     4     0     0     0     0     0
↦      0     0     0     0    -1     5     0     0     0     0
↦      0     0     0    -1     0     0     4     0     0     0
↦     -1     0     0     0     0     0     0     1     0     0
↦     -1     0     0     0     0     0     0    -1     2     0
↦      0     0     0     0     0     0     0     0    -1     3
/// If M is the matrix of multiplicities and TM the matrix of total
/// multiplicities of the singularity, then  M=P*TM.
/// We therefore calculate the (total) multiplicities. Note that
/// they have to be slightly extended.
list MULT=extend_multiplicities(totalmultiplicities(f1*f2*f3));
intmat TM=MULT[1];  // Total multiplicities.
intmat M=MULT[2];   // Multiplicities.
/// Check: M-P*TM=0.
M-proximity[1]*TM;
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0
/// Check: inverse(P)*M-TM=0.
intmat_inverse(proximity[1])*M-TM;
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0,
↦ 0,0,0
```

See also: Section D.6.1.3 [alexanderpolynomial], page 1628; Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.6 multseq2charexp

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Assume: The input is an intvec, which contains the mutiplicity sequence of an irreducible plane curve singularity .

Return: An intvec, which contains the sequence of characteristic exponents of the irreducible plane curve singularity defined by v.

Example:

```
LIB "alexpoly.lib";
intvec v=2,1,1;
multseq2charexp(v);
↦ 2,3
intvec v1=4,2,2,1,1;
multseq2charexp(v1);
↦ 4,6,7
```

D.6.1.7 charexp2multseq

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: charexp2multseq(v), v intvec

Assume: v contains the characteristic exponents of an irreducible plane curve singularity

Return: intvec, the multiplicity sequence of the plane curve singularity

Note: If the curve singularity is smooth, then the multiplicity sequence is empty. This is expressed by returning zero.

Example:

```
LIB "alexpoly.lib";
charexp2multseq(intvec(28,64,66,77));
↦ 28,28,8,8,8,4,4,2,2,2,2,2,2,2,1,1
```

See also: Section D.6.1.3 [alexanderpolynomial], page 1628; Section D.6.15.6 [invariants], page 1726; Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.8 charexp2generators

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: charexp2generators(v), v intvec

Assume: v contains the characteristic exponents of an irreducible plane curve singularity

Return: intvec, the minimal set of generators of the semigroup of the plane curve singularity

Example:

```
LIB "alexpoly.lib";
charexp2generators(intvec(28,64,66,77));
↦ 28,64,450,911
```

See also: Section D.6.1.3 [alexanderpolynomial], page 1628; Section D.6.15.6 [invariants], page 1726; Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.9 charexp2inter

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: charexp2inter(contact,charexp), contact matrix, charexp list

Assume: charexp contains the integer vectors of characteristic exponents of the branches of a plane curve singularity, and contact is their contact matrix

Return: intmat, the matrix intersection multiplicities of the branches

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ds;
list INV=invariants((x2-y3)*(x3-y2)*((x2-y3)^2-4x5y-x7));
intmat contact=INV[4][1];
list charexp=INV[1][1],INV[2][1],INV[3][1];
// The intersection matrix is INV[4][2].
print(INV[4][2]);
↦      0     4     8
↦      4     0    17
↦      8    17     0
// And it is calculated as ...
print(charexp2inter(contact,charexp));
↦      0     4     8
↦      4     0    17
↦      8    17     0
```

See also: Section D.6.15.6 [invariants], page 1726; Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.4 [semigroup], page 1629; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.10 charexp2conductor

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Assume: v contains the characteristic exponents of an irreducible plane curve singularity

Return: int, the conductor of the plane curve singularity

Note: If the curve singularity is smooth, the conductor is zero.

Example:

```
LIB "alexpoly.lib";
charexp2conductor(intvec(2,3));  // A1-Singularity
↦ 2
charexp2conductor(intvec(28,64,66,77));
↦ 1718
```

See also: Section D.6.15.6 [invariants], page 1726; Section D.6.1.1 [resolutiongraph], page 1626; Section D.6.1.4 [semigroup], page 1629; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.1.11 charexp2poly

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Assume: v an intvec containing the characterictic exponents of an irreducible plane curve singularity. a a vector containing the coefficients of a parametrization given by x(t)=x^v[1], y(t)=a(1)t^v[2]+...+a[n-1]t^v[n], i.e. the entries of a are of type number.

Return: A polynomial f in the first two variables of the basering, such that f defines an irreducible plane curve singularity with characteristic exponents v.

Note: The entries in a should be of type number and the vector v should be the sequence of characteristic exponents of an irreducible plane curve singularity in order to get a sensible result,

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),dp;
intvec v=8,12,14,17;
vector a=[1,1,1];
poly f=charexp2poly(v,a);
f;
↦ -x17+8x16-20x15+17x14-16x13y+12x12y2-2x13+32x12y-16x11y2-8x10y3+x12-8x11y\
   +20x10y2-16x9y3-4x9y2+16x8y3-2x7y4-8x6y5+6x6y4-8x5y5-4x3y6+y8
invariants(f)[1][1];  // The characteristic exponents of f.
↦ 8,12,14,17
```

See also: Section D.6.1.7 [charexp2multseq], page 1633; Section D.6.1.6 [multseq2charexp], page 1633.

D.6.1.12 tau_es2

Procedure from library `alexpoly.lib` (see Section D.6.1 [alexpoly_lib], page 1626).

Usage: tau_es2(INPUT); INPUT poly or list

Assume: INPUT is either a REDUCED bivariate polynomial defining a plane curve singularity, or the output of `hnexpansion(f[,"ess"])`, or the list `hne` in the ring created by hnexpansion(f[,"ess"]), or the output of develop(f) resp. of extdevelop(f,n), or a list containing the contact matrix and a list of integer vectors with the characteristic exponents of the branches of a plane curve singularity, or an integer vector containing the characteristic exponents of an irreducible plane curve singularity.

Return: int, the equisingular Tjurina number of f, i. e. the codimension of the mu-constant stratum in the semiuniversal deformation of f, where mu is the Milnor number of f.

Note: The equisingular Tjurina number is calculated with the aid of a Hamburger-Noether expansion, which is the hard part of the calculation.
In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of hnexpansion and use it as input instead of the polynomial itself.
If you are not sure whether the INPUT polynomial is reduced or not, use squarefree(INPUT) as input instead.

Example:

```
LIB "alexpoly.lib";
ring r=0,(x,y),ls;
poly f1=y2-x3;
poly f2=(y2-x3)^2-4x5y-x7;
poly f3=y3-x2;
tau_es2(f1);
↦ 2
tau_es2(f2);
↦ 14
tau_es2(f1*f2*f3);
↦ 49
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.12 [equising_lib], page 1696; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.12.1 [tau_es], page 1697; Section D.6.1.2 [totalmultiplicities], page 1627.

D.6.2 arcpoint_lib

Library: arcpoint.lib

Purpose: Truncations of arcs at a singular point

Author: Nadine Cremer cremer@mathematik.uni-kl.de

Overview: An arc is given by a power series in one variable, say t, and truncating it at a positive integer i means cutting the t-powers > i. The set of arcs truncated at order <bound> is denoted Tr(i). An algorithm for computing these sets (which happen to be constructible) is given in [Lejeune-Jalabert, M.: Courbes trac'ees sur un germe d'hypersurface, American Journal of Mathematics, 112 (1990)]. Our procedures for computing the locally closed sets contributing to the set of truncations rely on this algorithm.

Procedures:

D.6.2.1 nashmult

Procedure from library arcpoint.lib (see Section D.6.2 [arcpoint_lib], page 1636).

Usage: nashmult(f,bound); f polynomial, bound positive integer

Create: allsteps:

a list containing all relevant locally closed sets
up to order <bound> and their sequences of
Nash Multiplicities

setstep:

list of relevant locally closed sets
obtained from sequences of length bound+1

Return: ring, original basering with additional
variables t and coefficients up to t^<bound>

Example:

```
LIB "arcpoint.lib";
ring r=0,(x,y,z),dp;
poly f=z4+y3-x2;
def R=nashmult(f,2);
setring R;
allsteps;
↦ [1]:
↦    [1]:
↦       [1]:
↦          2,2
↦       [2]:
↦          _[1]=a(1)
↦          _[2]=b(1)
↦       [3]:
↦          _[1]=1
↦ [2]:
↦    [1]:
↦       [1]:
↦          2,2,1
↦       [2]:
↦          _[1]=a(1)
↦          _[2]=b(1)
↦          _[3]=c(1)^4-a(2)^2
↦       [3]:
↦          _[1]=a(1)
↦          _[2]=b(1)
↦          _[3]=c(1)
↦          _[4]=a(2)
↦    [2]:
↦       [1]:
↦          2,2,2
↦       [2]:
↦          _[1]=a(1)
↦          _[2]=b(1)
↦          _[3]=c(1)
↦          _[4]=a(2)
↦       [3]:
↦          _[1]=1
```

D.6.2.2 removepower

Procedure from library `arcpoint.lib` (see Section D.6.2 [arcpoint_lib], page 1636).

Usage: removepower(I); I ideal

Return: ideal defining the same zeroset as I: if any generator of I is a power of one single variable, replace it by the respective variable

Example:

```
LIB "arcpoint.lib";
ring r=0,(x,y,z),dp;
ideal I = x3,y+z2-x2;
I;
↦ I[1]=x3
↦ I[2]=-x2+z2+y
removepower(I);
↦ _[1]=x
↦ _[2]=-x2+z2+y
```

See also: Section D.6.2.3 [idealsimplify], page 1638.

D.6.2.3 idealsimplify

Procedure from library `arcpoint.lib` (see Section D.6.2 [arcpoint_lib], page 1636).

Usage: idealsimplify(I,m); I ideal, m int

Assume: procedure is stable for sufficiently large m

Return: ideal defining the same zeroset as I: replace generators of I by the generator modulo other generating elements

Example:

```
LIB "arcpoint.lib";
ring r=0,(x,y,z),dp;
ideal I = x3,y+z2-x2;
I;
↦ I[1]=x3
↦ I[2]=-x2+z2+y
idealsimplify(I,10);
↦ _[1]=x
↦ _[2]=z2+y
```

D.6.2.4 equalJinI

Procedure from library `arcpoint.lib` (see Section D.6.2 [arcpoint_lib], page 1636).

Usage: equalJinI(I,J); (I,J ideals)

Assume: J contained in I and both I and J have been processed with idealsimplify before

Return: 1, if I=J, 0 otherwise

Example:

```
LIB "arcpoint.lib";
ring r=0,(x,y,z),dp;
ideal I = x,y+z2;
ideal J1= x;
ideal J2= x,y+z2;
equalJinI(I,J1);
↦ 0
```

```
equalJinI(I,J2);
↦ 1
```

See also: Section D.6.2.3 [idealsimplify], page 1638.

D.6.3 arnoldclassify_lib

Library: arnoldClassify.lib

Purpose: Arnol'd Classifier of Singularities

Author: Eva Maria Hemmerling, ehemmerl@rhrk.uni-kl.de

Overview: A library for classifying isolated hypersurface singularities from the list of V.I. Arnol'd w.r.t. right equivalence up to corank 2. The method relies on Baciu's list of Milnor codes and Newton's rotating ruler method to distinguish the Y- and Z- singularities.

References:

[AVG85] Arnold, Varchenko, Gusein-Zade: Singularities of Differentiable Maps. Vol. 1: The classification of critical points caustics and wave fronts. Birkh"auser, Boston 1985

[Bac01] Corina Baciu: The classification of Hypersurface Singularities using the Milnor Code, Diplomarbeit, Universit"at Kaiserslautern, 2001.

[GP12] Greuel, Pfister: A SINGULAR Introduction to Commutative Algebra, Springer Science and Business Media, 2012

[Hem17] Eva Maria Hemmerling: Algorithmic Arnol'd Classification in SINGULAR, Master Thesis, TU Kaiserslautern, 2017.

Procedures: See also: Section D.6.4 [classify_lib], page 1645; Section D.6.19 [realclassify_lib], page 1745.

D.6.3.1 arnoldListAllSeries

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldListAllSeries();

Retrurn: list of names of singularity series listed by Arnol'd up to corank 2

Example:

```
LIB "arnoldclassify.lib";
arnoldListAllSeries();
↦ [1]:
↦    A[k]
↦ [2]:
↦    D[k]
↦ [3]:
↦    E[6k]
↦ [4]:
↦    E[6k+1]
↦ [5]:
↦    E[6k+2]
↦ [6]:
↦    J[k,0]
↦ [7]:
↦    J[k,r]
↦ [8]:
↦    W[12k]
↦ [9]:
↦    W[12k+1]
↦ [10]:
↦    W[12k+5]
↦ [11]:
↦    W[12k+6]
↦ [12]:
↦    W[k,0]
↦ [13]:
↦    W[k,r]
↦ [14]:
↦    W#[k,2r]
↦ [15]:
↦    W#[k,2r-1]
↦ [16]:
↦    X[k,0]
↦ [17]:
↦    X[k,r]
↦ [18]:
↦    Y[k,r,s]
↦ [19]:
↦    Z[k,r]
↦ [20]:
↦    Z[k,r,s]
↦ [21]:
↦    Z[1,6r+11]
↦ [22]:
↦    Z[1,6r+12]
↦ [23]:
↦    Z[1,6r+13]
↦ [24]:
↦    Z[k,12k+6r]
↦ [25]:
↦    Z[k,12k+6r+1]
↦ [26]:
↦    Z[k,12k+6r-1]
```

D.6.3.2 arnoldShowSeries

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldShowSeries(S); S string

Assume: S is the name of a singularity series listed by arnoldListAllSeries().

Return: data of the singularity series S of type singseries including
- Milnor number of S,
- Corank of S,
- Milnor code of S (see [Bac01]),
- normal form of S as string with parameters k,r,s and a,b,c,d,
- restrictions on parameters in the normal form in SINGULAR syntax,
- normal form with special (valid) parameters.

Example:

```
LIB "arnoldclassify.lib";
arnoldShowSeries("Z[k,12k+6r]");
↦ Series=Z[k,12k+6r]
↦ NormalForm=(x + a(y)*y^k)*(x^3 + x*y^(2*k+2*r+1) +
↦         b(y)* y^(3*k+3*r+2))
↦ SpecialForm=(x + y^k)*(x^3 + x*y^(2*k+2*r+1) +y^(3*k+3*r+2))
↦ Modality=3*k+r-2
↦ Corank=2
↦ MilnorNumber=12*k+6r
↦ MilnorCode=1,1,2k-1,2k-1+2*r,2k+2*r
↦ Restrictions= (k>1)&&(r>=0)&&(deg(a)<=(k-2))&&(jet(a,0)!=0)&&
↦         (jet(b,0)!=0)&&(deg(b)<=(2*k+r-2))
```

D.6.3.3 arnoldNormalForm

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldNormalForm(S [, l]), S string or singclass, l list

Assume: If S is of type string, then S is the name of a singularity series as listed by arnoldListAllSeries() and l may contain suitable integer parameters k,r,s. Otherwise S of type singclass is a singularity class from Arnol'd's list.
Optional suitable polynomial parameters a,b,c,d can be appended to l. If a,b,c,d are not given, valid values are chosen automatically.

Return: string NF is the normal form of the series S if no parameters given, or poly NF is the normal form of the class S with parameters k,r,s.

Example:

```
LIB "arnoldclassify.lib";
ring R = 0, (x,y), ds;
poly a,b,c,d;
a= 1+y2;
c= 3+y;
int k = 5;
int r = 4;
int s = 3;
arnoldNormalForm ("W[12k+1]", k,r,s,a,b,c,d);
↦ x4+x2y11+x2y13+xy16+3y22+y23
def f = _;
def sf = arnoldClassify( f );
arnoldNormalForm(sf, a,b,c,d);
↦ x4+x2y11+x2y13+xy16+3y22+y23
arnoldNormalForm("W[12k+1]");
↦ x4+x*y^(3*k+1)+y^(4*k+2)
arnoldNormalForm(sf);
↦ x4+xy16+y22
```

D.6.3.4 arnoldClassify

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldClassify (f); f poly

Assume: The basering is local of characteristic 0 and f defines an isolated singularity from Arnol'd's list of corank at most 2.

Compute: singularity class with respect to right equivalence and invariants used in the process of classification

Return: Singularity class of f of type singclass containing
- name of singularity series as listed by arnoldListAllSeries(),
- name of singularity class,
- parameters k,r,s defining the singularity class, -1 if not used,
- modality, corank, Milnor number, determinacy,
- Tjurina number, -2 if not computed, -1 if infinite,
- Milnor code, -1 if not computed,
- normal form of the singularity series from Arnol'd's list,
- restrictions on parameters as string in SINGULAR syntax.

Example:

```
LIB "arnoldclassify.lib";
ring r = 0,(x,y),ds;
int k = random(3,10);
poly g = x4 + x2*y^(2*k+1)+x*y^(3*k+1)+ y^(4*k +1);
arnoldClassify(g);
↦ Series=W[12k]
↦ Class=W[60]
↦ k=5
↦ r=-1
↦ s=-1
↦ Modality=13
↦ Corank=2
↦ Milnor=60
↦ Determinacy=29
↦ Tjurina=-2
↦ MilnorCode=1,1,10,9,9
↦ NormalForm=x4+a(y)*x*y^(3*k+1)+c(y)*x^2*y^(2*k+1)+y^(4*k+1)
↦ Restrictions=(k>=1)&&(k>1||a==0)&&(deg(a)<=(k-2))&&
↦          (deg(c)<=(2*k-2))
map phi=r,y-x^2,x+y;
phi(g);
↦ y4-4x2y3+6x4y2-4x6y+x8+x11y2+11x10y3+55x9y4+165x8y5+330x7y6+462x6y7+462x5\
   y8+330x4y9+165x3y10+55x2y11+11xy12+y13-2x13y-22x12y2-110x11y3-330x10y4-66\
   0x9y5-924x8y6-924x7y7-660x6y8-330x5y9-110x4y10-22x3y11-2x2y12+x15+11x14y+\
   55x13y2+165x12y3+330x11y4+462x10y5+462x9y6+330x8y7+165x7y8+55x6y9+11x5y10\
   +x4y11+x16y+16x15y2+120x14y3+560x13y4+1820x12y5+4368x11y6+8008x10y7+11440\
   x9y8+12870x8y9+11440x7y10+8008x6y11+4368x5y12+1820x4y13+560x3y14+120x2y15\
   +16xy16+y17-x18-16x17y-120x16y2-560x15y3-1820x14y4-4368x13y5-8008x12y6-11\
   440x11y7-12870x10y8-11440x9y9-8008x8y10-4368x7y11-1820x6y12-560x5y13-120x\
   4y14-16x3y15-x2y16+x21+21x20y+210x19y2+1330x18y3+5985x17y4+20349x16y5+542\
   64x15y6+116280x14y7+203490x13y8+293930x12y9+352716x11y10+352716x10y11+293\
   930x9y12+203490x8y13+116280x7y14+54264x6y15+20349x5y16+5985x4y17+1330x3y1\
   8+210x2y19+21xy20+y21
arnoldClassify(phi(g));
↦ Series=W[12k]
↦ Class=W[60]
↦ k=5
↦ r=-1
↦ s=-1
↦ Modality=13
↦ Corank=2
↦ Milnor=60
↦ Determinacy=29
↦ Tjurina=-2
↦ MilnorCode=1,1,10,9,9
↦ NormalForm=x4+a(y)*x*y^(3*k+1)+c(y)*x^2*y^(2*k+1)+y^(4*k+1)
↦ Restrictions=(k>=1)&&(k>1||a==0)&&(deg(a)<=(k-2))&&
↦         (deg(c)<=(2*k-2))
ring C = (0,i), (x,y), ds;
minpoly = i2 + 1;
poly f =(x2+y2)^2+x5;
arnoldClassify(f);
↦ Series=Y[1,r,s]
↦ Class=Y[1,1,1]
↦ k=1
↦ r=1
↦ s=1
↦ Modality=1
↦ Corank=2
↦ Milnor=11
↦ Determinacy=5
↦ Tjurina=-2
↦ MilnorCode=1,1,1,2,1
↦ NormalForm= x^(4+r)+ a(y)*x2*y2 + y^(4+s)
↦ Restrictions=(deg(a)==0)&&(jet(a,0)!=0)&&(1<=s)&&(s<=r)
```

D.6.3.5 arnoldClassify_to_string

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldClassify_to_string (f); f poly

Assume: The basering is local of characteristic 0 and f defines an isolated singularity from Arnol'd's list of corank at most 2.

Compute: singularity class with respect to right equivalence and invariants used in the process of classification

Return: string: separated by |:
- name of singularity series as listed by arnoldListAllSeries(),
- name of singularity class,
- parameters k,r,s defining the singularity class, -1 if not used,
- modality, corank, Milnor number, determinacy,
- Tjurina number, -2 if not computed, -1 if infinite,
- Milnor code, -1 if not computed,
- normal form of the singularity series from Arnol'd's list,
- restrictions on parameters as string in SINGULAR syntax.

Example:

```
LIB "arnoldclassify.lib";
ring r = 0,(x,y),ds;
int k = random(3,10);
```

```
poly g = x4 + x2*y^(2*k+1)+x*y^(3*k+1)+ y^(4*k +1);
arnoldClassify_to_string(g);
↦ (k>=1)&&(k>1||a==0)&&(deg(a)<=(k-2))&&
↦         (deg(c)<=(2*k-2))|x4+a(y)*x*y^(3*k+1)+c(y)*x^2*y^(2*k+1)+y^(4*k+1\
   )|1,1,10,9,9|-2|29|60|2|13|-1|-1|W[60]|W[12k]
```

D.6.3.6 arnoldCorank

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldCorank(f); f poly

Assume: basering is local, f in maxideal(2) has isolated critical point at 0

Return: corank of the Hessian matrix of f

Example:

```
LIB "arnoldclassify.lib";
ring r=0,(x,y,z),ds;
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
arnoldCorank(f);
↦ 2
```

D.6.3.7 arnoldDeterminacy

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldDeterminacy(I[, m]); I poly or ideal, m int.

Assume: the basering is local, I is the Jacobian ideal of a polynomial f with isolated critical point at 0, m is the Milnor number of f

Compute: determinacy bound k for f w.r.t. right equivalence

Return: integer k s.th. f is right-k-determined, -1 for infinity

Note: uses [Cor. A.9.7,GP12]

Example:

```
LIB "arnoldclassify.lib";
ring r=0,(x,y),ds;
poly f=x3+xy3;
ideal I=std(jacob(f));
int k=arnoldDeterminacy(I);
print(k);
↦ 5
```

D.6.3.8 arnoldMilnorCode

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldMilnorCode(f[,e]); f poly, e int

Assume: basering is local, f has isolated critical point at 0

Compute: Milnor code of f consisting of the numbers of successive repetitions of coefficients of the 2nd Hilbert series of basering/(jacob(f)^e), see [Bac01].

Return: Milnor code of f as intvec where e=1 by default

Example:

```
LIB "arnoldclassify.lib";
ring r=0,(x,y,z),ds;
poly f=x2y+y3+z2;
arnoldMilnorCode(f);
↦ 1,1,1
arnoldMilnorCode(f,2);
↦ 1,1,2,1
// a big second argument may result in memory overflow
```

D.6.3.9 arnoldMorseSplit

Procedure from library `arnoldclassify.lib` (see Section D.6.3 [arnoldclassify_lib], page 1639).

Usage: arnoldMorseSplit(f); f poly

Assume: base ring is local, f in maxideal(2) has isolated critical point at 0

Compute: result of Splitting Lemma applied to f

Return: polynomial g in maxideal(3) right equivalent to f

Example:

```
LIB "arnoldclassify.lib";
ring r=0,(x,y,z),ds;
export r;
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
poly g=arnoldMorseSplit(f);
g;
↦ 3/2x2y-y3
```

D.6.4 classify_lib

Library: classify.lib

Purpose: Arnold Classifier of Singularities

Author: Kai Krueger, krueger@mathematik.uni-kl.de

Overview: A library for classifying isolated hypersurface singularities w.r.t. right equivalence, based on the determinator of singularities by V.I. Arnold.

Procedures: See also: Section D.6.19 [realclassify_lib], page 1745.

D.6.4.1 basicinvariants

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: basicinvariants(f); f = poly

Compute: Compute basic invariants of f: an upper bound d for the determinacy, the milnor number mu and the corank c of f

Return: intvec: d, mu, c

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
basicinvariants((x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3);
↦ 5,4,2
```

D.6.4.2 classify

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: classify(f); f=poly

Compute: normal form and singularity type of f with respect to right equivalence, as given in the book "Singularities of differentiable maps, Volume I" by V.I. Arnold, S.M. Gusein-Zade, A.N. Varchenko

Return: normal form of f, of type poly

Remark: This version of classify is only beta. Please send bugs and comments to: "Kai Krueger" <krueger@mathematik.uni-kl.de>
Be sure to have at least Singular version 1.0.1.

Note: type init_debug(n); (0 <= n <= 10) in order to get intermediate information, higher values of n give more information.
The proc creates several global objects with names all starting with @, hence there should be no name conflicts.

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
classify(f);
↦ About the singularity :
↦           Milnor number(f)   = 4
↦           Corank(f)          = 2
↦           Determinacy       <= 5
↦ Guessing type via Milnorcode:   D[k]=D[4]
↦ 
↦ Computing normal form ...
↦ I have to apply the splitting lemma. This will take some time....:-)
↦    Arnold step number 4
↦ The singularity
↦    -x3+3/2xy2+1/2x3y-1/16x2y2+3x2y3
↦ is R-equivalent to D[4].
↦    Milnor number = 4
↦    modality      = 0
↦ 2z2+x2y+y3
init_debug(3);
↦ Debugging level change from  0  to  3
classify(f);
↦ Computing Basicinvariants of f ...
↦ About the singularity :
↦           Milnor number(f)   = 4
↦           Corank(f)          = 2
↦           Determinacy       <= 5
↦ Hcode: 1,2,1,0,0
↦ Milnor code :  1,1,1
↦ Debug:(2):  entering HKclass3_teil_1 1,1,1
↦ Debug:(2):  finishing HKclass3_teil_1
↦ Guessing type via Milnorcode:   D[k]=D[4]
↦ 
↦ Computing normal form ...
↦ I have to apply the splitting lemma. This will take some time....:-)
↦ Debug:(3):  Split the polynomial below using determinacy:  5
↦ Debug:(3):  9y2-12yz+4z2-x3+6x2y-4x2z+xyz+x4+3x2y3
↦ Debug:(2):  Permutations: 3,2,1
↦ Debug:(2):  Permutations: 3,2,1
↦ Debug:(2):  rank determined with Morse rg= 1
↦ Residual singularity f= -x3+3/2xy2+1/2x3y-1/16x2y2+3x2y3
↦ Step 3
↦    Arnold step number 4
↦ The singularity
↦    -x3+3/2xy2+1/2x3y-1/16x2y2+3x2y3
↦ is R-equivalent to D[4].
↦    Milnor number = 4
↦    modality      = 0
↦ Debug:(2):  Decode:
↦ Debug:(2):  S_in= D[4]   s_in= D[4]
↦ Debug:(2):  Looking for Normalform of  D[k] with (k,r,s) = ( 4 , 0 , 0 )
↦ Debug:(2):  Opening Singalarity-database:
↦  DBM: NFlist
↦ Debug:(2):  DBMread( D[k] )= x2y+y^(k-1) .
↦ Debug:(2):  S= f = x2y+y^(k-1);  Tp= x2y+y^(k-1) Key= I_D[k]
↦ Polynom f= x2y+y3   crk= 2   Mu= 4  MlnCd= 1,1,1
↦ Debug:(2):  Info= x2y+y3
↦ Debug:(2):  Normal form NF(f)= 2*x(3)^2+x(1)^2*x(2)+x(2)^3
↦ 2z2+x2y+y3
```

D.6.4.3 corank

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: corank(f); f=poly

Return: the corank of the Hessian matrix of f, of type int

Remark: corank(f) is the number of variables occurring in the residual singularity after applying 'morsesplit' to f

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
corank(f);
↦ 2
```

D.6.4.4 Hcode

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: Hcode(v); v=intvec

Return: intvec, coding v according to the number of successive repetitions of an entry

Example:

```
LIB "classify.lib";
intvec v1 = 1,3,5,5,2;
Hcode(v1);
```

```
↦ 1,0,1,0,2,0,0,1,0
intvec v2 = 1,2,3,4,4,4,4,4,4,4,3,2,1;
Hcode(v2);
↦ 1,1,1,7,1,1,1
```

D.6.4.5 init_debug

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: init_debug([level]); level=int

Compute: Set the global variable @DeBug to level. The variable @DeBug is used by the function debug_log(level, list of strings) to know when to print the list of strings. init_debug() reports only changes of @DeBug.

Note: The procedure init_debug(n); is useful as trace-mode. n may range from 0 to 10, higher values of n give more information.

Example:

```
LIB "classify.lib";
init_debug();
debug_log(1,"no trace information printed");
init_debug(1);
↦ Debugging level change from  0  to  1
debug_log(1,"some trace information");
↦ some trace information
init_debug(2);
↦ Debugging level change from  1  to  2
debug_log(2,"nice for debugging scripts");
↦ Debug:(2):  nice for debugging scripts
init_debug(0);
↦ Debugging switched off.
```

D.6.4.6 internalfunctions

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: internalfunctions();

Return: nothing, display names of internal procedures of classify.lib

Example:

```
LIB "classify.lib";
internalfunctions();
↦    Internal functions for the classification using Arnold's method,
↦    the function numbers correspond to numbers in Arnold's classifier:
↦ Klassifiziere(poly f);      //determine the type of the singularity f
↦   Funktion1bis (poly f, list cstn)
↦   Funktion3 (poly f, list cstn)
↦   Funktion6 (poly f, list cstn)
↦   Funktion13 (poly f, list cstn)
↦   Funktion17 (poly f, list cstn)
↦   Funktion25 (poly f, list cstn)
↦   Funktion40 (poly f, list cstn, int k)
↦   Funktion47 (poly f, list cstn)
↦   Funktion50 (poly f, list cstn)
↦   Funktion58 (poly fin, list cstn)
↦   Funktion59 (poly f, list cstn)
↦   Funktion66 (poly f, list cstn)
↦   Funktion82 (poly f, list cstn)
↦   Funktion83 (poly f, list cstn)
↦   Funktion91 (poly f, list cstn, int k)
↦   Funktion92 (poly f, list cstn, int k)
↦   Funktion93 (poly f, list cstn, int k)
↦   Funktion94 (poly f, list cstn, int k)
↦   Funktion95 (poly f, list cstn, int k)
↦   Funktion96 (poly f, list cstn, int k)
↦   Funktion97 (poly f, list cstn)
↦   Isomorphie_s82_x (poly f, poly fk, int k)
↦   Isomorphie_s82_z (poly f, poly fk, int k)
↦   Isomorphie_s17 (poly f, poly fk, int k, int ct)
↦   printresult (string f,string typ,int Mu,int m,int corank,int K)
↦
↦    Internal functions for the classifcation by invariants:
↦   Cubic (poly f)
↦   parity (int e)             //return the parity of e
↦   HKclass (intvec i)
↦   HKclass3( intvec i, string SG_Typ, int cnt)
↦   HKclass3_teil_1 (intvec i, string SG_Typ, int cnt)
↦   HKclass5 (intvec i, string SG_Typ, int cnt)
↦   HKclass5_teil_1 (intvec i, string SG_Typ, int cnt)
↦   HKclass5_teil_2 (intvec i, string SG_Typ, int cnt)
↦   HKclass7 (intvec i, string SG_Typ, int cnt)
↦   HKclass7_teil_1 (intvec i, string SG_Typ, int cnt)
↦
↦    Internal functions for the Morse-splitting lemma:
↦   Morse(poly fi, int K, int corank)  //splitting lemma itself
↦   Coeffs (list #)
↦   Coeff
↦
↦    Internal functions providing tools:
↦   ReOrder(poly f)
↦   Singularitaet(string typ,int k,int r,int s,poly a,poly b,poly c,poly d)
↦   RandomPolyK
↦   Faktorisiere(poly f, poly g, int p, int k)   compute g = (ax+by^k)^p
↦   Teile(poly f, poly g);              //divides f by g
↦   GetRf(poly f, int n);
↦   Show(poly f);
↦   checkring();
↦   DecodeNormalFormString(string s);
↦   Setring(int n, string ringname);
↦
```

D.6.4.7 milnorcode

Procedure from library `classify.lib` (see Section D.6.4 [classify lib], page 1645).

Usage: milnorcode(f[,e]); f=poly, e=int

Return: intvec, coding the Hilbert function of the e-th Milnor algebra of f, i.e. of basering/(jacob(f)^e) (default e=1), according to proc Hcode

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
poly f=x2y+y3+z2;
milnorcode(f);
↦ 1,1,1
milnorcode(f,2);  // a big second argument may result in memory overflow
↦ 1,0,1,0,2,0,0,1,0
```

D.6.4.8 morsesplit

Procedure from library classify.lib (see Section D.6.4 [classify_lib], page 1645).

Usage: morsesplit(f); f=poly

Return: Normal form of f in M^3 after application of the splitting lemma

Compute: apply the splitting lemma (generalized Morse lemma) to f

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
export r;
init_debug(1);
↦ Debugging level is set to  1
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
poly g=morsesplit(f);
↦ Residual singularity f= -x3+3/2xy2+1/2x3y-1/16x2y2+3x2y3
g;
↦ -x3+3/2xy2+1/2x3y-1/16x2y2+3x2y3
```

D.6.4.9 quickclass

Procedure from library classify.lib (see Section D.6.4 [classify_lib], page 1645).

Usage: quickclass(f); f=poly

Return: Normal form of f in Arnold's list

Remark: try to determine the normal form of f by invariants, mainly by computing the Hilbert function of the Milnor algebra, no coordinate change is needed (see also proc 'milnorcode').

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),ds;
poly f=(x2+3y-2z)^2+xyz-(x-y3+x2*z3)^3;
quickclass(f);
↦ Singularity R-equivalent to :  D[k]=D[4]
↦ normal form : z2+x2y+y3
↦ z2+x2y+y3
```

D.6.4.10 singularity

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: singularity(t, l); t=string (name of singularity),
l=list of integers/polynomials (indices/parameters of singularity)

Compute: get the singularity named by type t from the database. list l is as follows:
l= k [,r [,s [,a [,b [,c [,d]..]: k,r,s=int a,b,c,d=poly.
The name of the dbm-databasefile is: NFlist.[dir,pag]. The file is found in the current directory. If it does not exist, please run the script MakeDBM first.

Return: Normal form and corank of the singularity named by type t and its index (indices) l.

Example:

```
LIB "classify.lib";
ring r=0,(x,y,z),(c,ds);
init_debug(0);
singularity("E[6k]",6);
↦ [1]:
↦    x3+xy13+y19
↦ [2]:
↦    2
singularity("T[k,r,s]", 3, 7, 5);
↦ [1]:
↦    x3+xyz+z5+y7
↦ [2]:
↦    3
poly f=y;
singularity("J[k,r]", 4, 0, 0, f);
↦ [1]:
↦    x3+x2y4+y13
↦ [2]:
↦    2
```

D.6.4.11 A_L

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: A_L(f); f poly
A_L(s); s string, the name of the singularity

Compute: the normal form of f in Arnold's list of singularities in case 1, in case 2 nothing has to be computed.

Return: A_L(f): compute via 'milnorcode' the class of f and return the normal form of f found in the database.
A_L("name"): get the normal form from the database for the singularity given by its name.

Example:

```
LIB "classify.lib";
ring r=0,(a,b,c),ds;
poly f=A_L("E[13]");
f;
↦ c2+a3+ab5+b8
```

```
A_L(f);
↦ Singularity R-equivalent to :  E[6k+1]=E[13]
↦ normal form : c2+a3+ab5+b8
↦ c2+a3+ab5+b8
```

D.6.4.12 normalform

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: normalform(s); s=string

Return: Arnold's normal form of singularity with name s

Example:

```
LIB "classify.lib";
ring r=0,(a,b,c),ds;
normalform("E[13]");
↦ c2+a3+ab5+b8
```

D.6.4.13 debug_log

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: debug_log(level,li); level=int, li=comma separated "message" list

Compute: print "messages" if level>=@DeBug.
useful for user-defined trace messages.

Example:

```
LIB "classify.lib";
example init_debug;
↦ // proc init_debug from lib classify.lib
↦ EXAMPLE:
↦   init_debug();
↦   debug_log(1,"no trace information printed");
↦   init_debug(1);
↦ Debugging level change from  0  to  1
↦   debug_log(1,"some trace information");
↦ some trace information
↦   init_debug(2);
↦ Debugging level change from  1  to  2
↦   debug_log(2,"nice for debugging scripts");
↦ Debug:(2):  nice for debugging scripts
↦   init_debug(0);
↦ Debugging switched off.
↦
```

See also: Section D.6.4.5 [init_debug], page 1648.

D.6.4.14 swap

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: swap(a,b);

Return: b,a if a,b is the input (any type)

Example:

```
LIB "classify.lib";
swap("variable1","variable2");
↦ variable2 variable1
```

D.6.4.15 modality

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: modality(f); f poly

Return: the modality of the singularity

Example:

```
LIB "classify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
modality(f);
↦ 0
```

D.6.4.16 complexSingType

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: complexSingType(f); f poly

Return: the type of the singularity as a string

Example:

```
LIB "classify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
complexSingType(f);
↦ D[4]
```

D.6.4.17 prepRealclassify

Procedure from library `classify.lib` (see Section D.6.4 [classify_lib], page 1645).

Usage: prepRealclassify(f); f poly

Return: a list, containing the modality of the singularity and the type of the singularity as a string
This procedure is needed in realclassify.lib in order to avoid classify() being called more than once.

Example:

```
LIB "classify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
prepRealclassify(f);
↦ [1]:
↦    0
↦ [2]:
↦    D[4]
```

D.6.5 classify2_lib

Library: classify2.lib

Purpose: Classification of isolated singularities

Authors: Janko Boehm, email: boehm@mathematik.uni-kl.de
Magdaleen Marais, email: magdaleen.marais@up.ac.za
Gerhard Pfister, email: pfister@mathematik.uni-kl.de

Overview: We classify isolated singularities of corank <=2 and modality <=2 with respect to right-equivalence over the complex numbers according to Arnold's list. We determine the type and, for positive modality, the parameter.

V.I. Arnold has described normal forms and has developed a classifier for, in particular, all isolated hypersurface singularities over the complex numbers up to modality 2. Building on a series of 105 theorems, this classifier determines the type of the given singularity. However, for positive modality, this does not fix the right equivalence class of the singularity, since the values of the moduli parameters are not specified.

This library implements an alternative classification algorithm for isolated hypersurface singularities of corank and modality up to two. For a singularity given by a polynomial over the rationals, the algorithm determines its right equivalence class by specifying a polynomial representative in Arnold's list of normal forms. In particular, the algorithm also determines values for the moduli parameters.

The implementation is based on the paper

Janko Boehm, Magdaleen Marais, Gerhard Pfister: A Classification Algorithm for Complex Singularities of Corank and Modality up to Two, Singularities and Computer Algebra - Festschrift for Gert-Martin Greuel on the Occasion of his 70th Birthday, Springer 2017, http://arxiv.org/abs/1604.04774, https://doi.org/10.1007/978-3-319-28829-1_2

There are functions for determining a normal form equation and for determining the complex type of the singularity.

Acknowledgements: This research was supported by
the Staff Exchange Bursary Programme of the University of Pretoria, DFG SPP 1489, DFG TRR 195. The financial assistance of the National Research Foundation (NRF), South Africa, towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the National Research Foundation, South Africa.

Procedures: See also: Section D.6.4 [classify_lib], page 1645; Section D.6.19 [realclassify_lib], page 1745.

D.6.5.1 complexClassify

Procedure from library `classify2.lib` (see Section D.6.5 [classify2_lib], page 1654).

Usage: complexClassify(fl); fl Poly

Assume: fl is a bivariate polynomial defining a curve singularity at (0,0) of modality <=2 and corank <=2. The ordering of the basering must be local.

Return: A normal form equation g for fl represented by an element of type NormalFormEquation.

Note: So far the final scaling step is not implemented, so to obtain a normal form equation it may be necessary to do a transformation of the form x->lambda1*x, y->lambda2*y

Note: Case X9 is not implemented yet. In this case an error is returned stating that it is case X9.

g.normalFormEquation.in stores the polynomial ring containing the normal form equation.

g.normalFormEquation.value is the normal form equation.

To access the normal form equation as a polynomial do:

def S = g.normalFormEquation.in;
setring S;
poly nf = g.normalFormEquation.value;

Example:

```
LIB "classify2.lib";
ring R = 0,(x,y),ds;
Poly f = -16065*y^9*x-5103*y^9-128*x^7-595*y^9*x^4+6*y^5-21*y^9*x^5-2187*y^7+4*x^2*y
NormalFormEquation F = complexClassify(f);
F;
↦ Corank = 2
↦ Normalform equation of type = Y[7,8]
↦ Normalform equation = x2y2-x7+y8
↦ Milnor number = 16
↦ Modality = 1
↦ Parameter term = x2y2
↦ Determinacy <= 8
↦
def S = F.normalFormEquation.in;
setring S;
poly nf = F.normalFormEquation.value;
```

D.6.5.2 complexType

Procedure from library `classify2.lib` (see Section D.6.5 [classify2_lib], page 1654).

Usage: complexType(fl); fl Poly

Assume: fl is a bivariate polynomial defining a curve singularity at (0,0) of modality and corank <=2. The ordering of the basering must be local.

Return: A list with the complex type of fl and the modality.

Example:

```
LIB "classify2.lib";
ring R = 0,(x,y),ds;
Poly f = -16065*y^9*x-5103*y^9-128*x^7-595*y^9*x^4+6*y^5-21*y^9*x^5-2187*y^7+4*x^2*y
complexType(f);
```

```
↦ [1]:
↦    Y[7,8]
↦ [2]:
↦    1
```

D.6.6 classify_aeq_lib

Library: classifyAeq.lib

Purpose: Simple Space Curve singularities in characteristic 0

Authors: Faira Kanwal Janjua fairakanwaljanjua@gmail.com
Gerhard Pfister pfister@mathematik.uni-kl.de Khawar Mehmood khawar1073@gmail.com

Overview: A library for classifying the simple singularities
with respect to A equivalence in characteristic 0.
Simple Surface singularities in characteristic O have been classified by Bruce and Gaffney [4] resp. Gibson and Hobbs [1] with respect to A equivalence. If the input is one of the simple singularities in [1] it returns a normal form otherwise a zero ideal(i.e not simple).

References:
[1] Gibson,C.G; Hobbs,C.A.:Simple SIngularities of Space Curves. Math.Proc. Comb.Phil.Soc.(1993),113,297.
[2] Hefez,A;Hernandes,M.E.:Standard bases for local rings of branches and their modules of differentials. Journal of Symbolic Computation 42(2007) 178-191. [3] Hefez,A;Hernandes,M.E.:The Analytic Classification Of Plane Branches. Bull.Lond Math Soc.43.(2011) 2,289-298. [4] Bruce, J.W.,Gaffney, T.J.: Simple singularities of mappings (C, 0) ->(C2,0). J. London Math. Soc. (2) 26 (1982), 465-474.
[5] Ishikawa,G; Janeczko,S.: The Complex Symplectic Moduli Spaces of Unimodal Parametric Plane Curve Singularities. Insitute of Mathematics of the Polish Academy of Sciences,Preprint 664(2006)

Procedures:

D.6.6.1 sagbiAlg

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: An ideal.The sagbi bases of I.

Example:

```
LIB "classify_aeq.lib";
ring R=0,t,ds;
ideal I=t8,t10+t13,t12+t15;
sagbiAlg(I);
↦ _[1]=t8
↦ _[2]=t10+t13
↦ _[3]=t12+t15
↦ _[4]=t23-t29
↦ _[5]=t27
I=t8,t10+t13,t12+2t15;
sagbiAlg(I);
↦ _[1]=t8
↦ _[2]=t10+t13
↦ _[3]=t12+2t15
↦ _[4]=t27-3t33
↦ _[5]=t29
```

D.6.6.2 sagbiMod

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: An ideal. the sagbi bases for the differential module.

Example:

```
LIB "classify_aeq.lib";
ring r=0,t,Ds;
ideal G=t8,t10+t13,t12+t15,t23-t29,t27;
ideal I=diff(G,t);
sagbiMod(I,G);
↦ _[1]=t7
↦ _[2]=t9+13/10t12
↦ _[3]=t11+5/4t14
↦ _[4]=t20
↦ _[5]=t22-29/23t28
↦ _[6]=t24
↦ _[7]=t26
↦ _[8]=t28
```

D.6.6.3 semiGroup

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: list L; list with three entries associated to the algebra generated by the sagbi basis:
generators of the semigroup
the conductor
the semigroup

Example:

```
LIB "classify_aeq.lib";
ring R=0,t,ds;
ideal I=t8,t10+t13,t12+t15,t23-t29,t27;
semiGroup(I);
↦ [1]:
↦    8,10,12,23,27
↦ [2]:
↦    30
↦ [3]:
↦    0,8,10,12,16,18,20,22,23,24,26,27,28,30
I=t8,t10+t13,t12+2t15,t27-3t33,t29;
semiGroup(I);
↦ [1]:
↦    8,10,12,27,29
↦ [2]:
↦    34
↦ [3]:
↦    0,8,10,12,16,18,20,22,24,26,27,28,29,30,32,34
```

D.6.6.4 semiMod

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: list K;
K[1]min generators of the semialgebra.
K[2]conductor of the algebra.
K[3]genrators for the semialgebra.
K[4]min generators of the module.
K[5]conductor of the module.
K[6]semigroup of the module.

Example:

```
LIB "classify_aeq.lib";
ring r=0,t,Ds;
ideal G=t4,t7+t10;
ideal I=diff(G,t);
ideal k=sagbiMod(I,G);
semiMod(k,G);
↦ [1]:
↦    4,7
↦ [2]:
↦    18
↦ [3]:
↦    0,4,7,8,11,12,14,15,16,18
↦ [4]:
↦    3,6,16
↦ [5]:
↦    13
↦ [6]:
↦    3,6,7,10,11,13
```

D.6.6.5 planeCur

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: An ideal.Ideal is one of the singularity in the list of Bruce and Gaffney [4]

Example:

```
LIB "classify_aeq.lib";
ring R=0,t,Ds;
ideal I=t4+4t5+6t6+8t7+13t8+12t9+10t10+12t11+6t12+4t13+4t14+t16,t7+7t8+22t9+51t10+113
+1828t18+2011t19+2165t20+2163t21+1982t22+1806t23+1491t24+1141t25+889t26+588t27+379t28
planeCur(I);
↦ // ** redefining K (                intvec q=4,7;list K;) classify_aeq.li\
   b::planeCur:155
↦ _[1]=t4
↦ _[2]=t7+t9
```

D.6.6.6 spaceCur

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: an ideal. Ideal is one of the singularity in the list of C.G.Gibson and C.A.Hobbs.

Example:

```
LIB "classify_aeq.lib";
ring R=0,t,Ds;
ideal I=t3+3t4+3t5+t6,t13+14t14+92t15+377t16+1079t17+2288t18+3718t19+4719t20+4719t21+
spaceCur(I);
↦ _[1]=t3
↦ _[2]=t13+t14
↦ _[3]=t17
```

D.6.6.7 HHnormalForm

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Usage: HHnormalForm(I), I=<x(t),y(t)> an ideal, # optional, a bound for the conductor

Compute: computes the Hefez-Hernandez normal form of the ideal I=<x(t),y(t)>

Return: a list L with 5 entries
L[1] the Hefei-Hernandez normal form of the ideal I=<x(t),y(t)> L[2] the semi group Gamma
L[3] the semi module Lambda
L[4] 0 if Lambda=Gamma or lambda=min(Lambda-Gamma) -n, the Zariski number
L[5] the integers i, i>lambda, i+n not in Lambda, 0 if this set is empty

Example:

```
LIB "classify_aeq.lib";
ring r=0,t,Ds;
ideal I=t6+3t7,t8+t13+t19;
HHnormalForm(I);
↦ [1]:
↦    _[1]=t6
↦    _[2]=t8-4t9+425/6t11-823/32t13
↦ [2]:
↦    0,6,8,12,14,16,18,20,22,24,25,26,28,30,31,32,33,34,36
↦ [3]:
↦    0,6,8,12,14,15,16,18,20
↦ [4]:
↦    9
↦ [5]:
↦    11,13
```

D.6.6.8 modSagbiAlg

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Usage: modSagbiAlg(I,i); I is an ideal, i an integer (optional).

Return: ideal, a sagbi basis for I using modular methods.

Purpose: Computes a sagbi basis for the ideal given by the generators in I using modular techniques.
If second argument is 0 then the result is not verified.

Example:

```
LIB "classify_aeq.lib";
ring R=0,t,ds;
ideal I=t8,t10+t13,t12+t15;
```

```
modSagbiAlg(I,0);
↦ _[1]=t8
↦ _[2]=t10+t13
↦ _[3]=t12+t15
↦ _[4]=t23-t29
↦ _[5]=t27
modSagbiAlg(I);
↦ _[1]=t8
↦ _[2]=t10+t13
↦ _[3]=t12+t15
↦ _[4]=t23-t29
↦ _[5]=t27
```

D.6.6.9 classSpaceCurve

Procedure from library `classify_aeq.lib` (see Section D.6.6 [classify_aeq_lib], page 1656).

Return: the normal form of I if I is simple, -1 if I is not simple

Example:

```
LIB "classify_aeq.lib";
ring R=31,t,ds;
ideal I=t3+6t4+13t5-13t6+10t7+2t8-6t9-10t10-15t11-6t12+8t13-2t14+t15+8t16,
t7+15t8+7t9-11t10-15t11-6t12+8t13-2t14+t15+8t16,
t10-11t11-6t12-t13+12t14+4t15-14t16+15t17-12t18+5t19+t20;
ideal J=classSpaceCurve(I);
J;
↦ J[1]=t3
↦ J[2]=t7+t8
↦ J[3]=t11
```

D.6.7 classifyceq_lib

Library: classifyCeq.lib

Purpose: simple hypersurface singularities in characteristic p > 0

Authors: Deeba Afzal deebafzal@gmail.com
Faira Kanwal Janjua fairakanwaljanjua@gmail.com

Overview: A library for classifying the simple singularities with respect to contact equivalence in charateristic p > 0 . Simple hypersurface singularities in charateristic p > 0 were classified by Greuel and Kroening [1] with respect to contact equivalence. The classifier we use has been proposed in [2].

References:

[1] Greuel, G.-M.; Kroening, H.: Simple singularities in positive characteristic. Math.Z. 203, 339-354 (1990).
[2] Afzal,D.;Binyamin,M.A.;Janjua,F.K.: On the classification of simple singularities in positive characteristic.

Procedures:

D.6.7.1 classifyCeq

Procedure from library `classifyceq.lib` (see Section D.6.7 [classifyceq_lib], page 1660).

Usage: classifyCeq(f); f poly

Return: string including the Tjurina number of f and its type in the classification of Greuel and Kroening

Example:

```
LIB "classifyceq.lib";
ring R=3,(x,y,z,u,v,w),ds;
classifyCeq(-x2+xy+y2+xz-yz-z2+w2+u3+v4);
↦ The given function defines a simple singularity.
↦ The tjurina number is 9.
↦ E^0[6]:x^3+y^4
```

D.6.8 classifyci_lib

Library: classifyci.lib

Purpose: Isolated complete intersection singularities in characteristic 0

Authors: Gerhard Pfister pfister@mathematik.uni-kl.de
Deeba Afzal deebafzal@gmail.com

Overview: A library for classifying isolated complete intersection singularities for the base field of characteristic 0 and for computing weierstrass semigroup of the space curve.Isolated complete intersection singularities were classified by M.Giusti [1] for the base field of characteristic 0. Algorithm for the semigroup of a space curve singularity is given in [2].

References:

[1] Giusti,M:Classification des singularities isolees simples d'intersections completes, C,R.Acad.Sci.Paris Ser.A-B 284(1977),167-169.
[2] Castellanos,A.,Castellanos,J.,2005:Algorithm for the semigroup of a space curve singularity. Semigroup Forum 70,44-66.

Procedures:

D.6.8.1 classifyicis

Procedure from library `classifyci.lib` (see Section D.6.8 [classifyci_lib], page 1661).

Usage: classifyicis(I); I ideal

Assume: I is given by two generators

Purpose: Check whether the ideal defines a complete intersection singularity or not

Return: String type in the classification of Giusti,M
or The given singularity is not simple

Example:

```
LIB "classifyci.lib";
ring R=0,(x,y,z),ds;
ideal I=x2+yz,xy+z4;
classifyicis(I);
```

```
↦ // ** dim(_) may be wrong because the mixed monomial ordering
↦ // ** dim(_) may be wrong because the mixed monomial ordering
↦ // ** dim(I) may be wrong because the mixed monomial ordering
↦ // ** dim(I) may be wrong because the mixed monomial ordering
↦ U_9:(x2+yz,xy+z4)
```

D.6.8.2 Semigroup

Procedure from library `classifyci.lib` (see Section D.6.8 [classifyci_lib], page 1661).

Usage: Semigroup(l); I is an ideal

Purpose: Computes the semigroup of the ideal I corresponding to each branch

Return: list of semigroup of ideal I corresponding to each branch

Example:

```
LIB "classifyci.lib";
ring R=0,(x,y,z),ds;
ideal I=x2+y3+z5,yz;
Semigroup(I);
↦ [1]:
↦    2,3
↦ [2]:
↦    2,5
```

D.6.9 classifyMapGerms_lib

Library: classifyMapGerms.lib

Authors: Gerhard Pfister, pfister@mathematik.uni-kl.de
Deeba Afzal, deebafzal@gmail.com
Shamsa Kanwal, lotus_zone16@yahoo.com

Overview: A library for computing the standard basis of the tangent space at the orbit of an algebraic group action. The tangent space is usually described as the sum of two modules over different rings. It computes the standard basis using modular methods and parallel modular methods. It also computes the normal form of the germ given by Riegers classification.

References:

[1] Idrees N.; Pfister, G.; Steidel, S.: Parallelization of modular algorithms. J. Symbolic Comput. 46(2011), no. 6, 672-684.
[2] Gibson,C.G; Hobbs,C.A.: Simple SIngularities of Space Curves. Math.Proc. Comb.Phil.Soc.(1993),113,297.
[3] Bruce, J.W.,Gaffney, T.J.: Simple singularities of mappings (C, 0) ->(C2,0). J. London Math. Soc. (2) 26 (1982), 465-474.
[4] Rieger, J. H.: Families of maps from the plane to the plane. J. London Math. Soc. (2)36(1987), no. 2. 351-369.

Procedures:

D.6.9.1 coDimMap

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: coDimMap(I, #); I=ideal, #=list

Compute: a bound of the A-determinacy of the map germ defined by I.

Return: a list in which 1st entry gives the bound of the A-determinancy and the second entry gives the codimension of the map germ defined by I.

Note: if # is empty it computes A^e-codimension(the extended codimension).

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
coDimMap(I);
↦ [1]:
↦    14
↦ [2]:
↦    2
coDimMap(I,1);
↦ [1]:
↦    15
↦ [2]:
↦    4
```

D.6.9.2 coDim

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: coDim(module M, module N, ideal I,int bound, list #); M is a submodule in A^r over the basering=:A, N is a submodule in R^r over the subring R of the basering generated by the entries of I

Compute: computes the K-vectorspace dimension of A^r/M+N+maxideal(bound)*A^r

Return: an integer

Note: if # is not empty the bound is corrected by computing determinacy

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
```

```
module M=maxideal(1)*jacob(I);
module N=I*freemodule(2);
coDim(M,N,I,15);
↦ 6
```

D.6.9.3 vStd

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: vStd(M, N, I, bound);M is a submodule in A^r over the basering=:A, N ist a submodule in R^r over the subring R of the basering generated by the entries of I

Compute: a standard basis of M+N+maxideal(bound)*A^r

Return: a list whose ist entry gives a list where each entry is a genrator of standard basis and second entry gives a list of generators after the reduced echelon form

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
module M=maxideal(1)*jacob(I);
module N=I*freemodule(2);
vStd(M,N,I,15);
↦ [1]:
↦    _[1]=[x+x3+4x2y+3xy2+2x3y+6x2y2+4xy3+5x2y4+6xy5+30x2y5+35xy6+77x2y6+88\
   xy7+136x2y7+153xy8+234x2y8+260xy9+360x2y9+396xy10+385x2y10+420xy11+252x2y\
   11+273xy12+91x2y12+98xy13+14x2y13+15xy14,3x2+4xy+2x2y+3xy2+5xy4+30xy5+77x\
   y6+136xy7+234xy8+360xy9+385xy10+252xy11+91xy12+14xy13]
↦    _[2]=[y+x2y+4xy2+3y3+2x2y2+6xy3+4y4+5xy5+6y6+30xy6+35y7+77xy7+88y8+136\
   xy8+153y9+234xy9+260y10+360xy10+396y11+385xy11+420y12+252xy12+273y13+91xy\
   13+98y14+14xy14+15y15,3xy+4y2+2xy2+3y3+5y5+30y6+77y7+136y8+234y9+360y10+3\
   85y11+252y12+91y13+14y14]
↦    _[3]=[0,362002419184362586664273749843544904887015830148107635007996401\
   21746364661822045 2x2+80358560767929054373410082522587858998935439927001270\
   487928566235908859788 3320904xy+4415831884949279570698270753823336851023381\
   56912190511378486022141624951265100452y2-121685417352384126148341273709341\
   226633725947951858159878722581143915558569733 12x3-8012829055622910724047\
   609502872071361548598882425479340633403208939604546316 4633x2y-15945044934\
   8534379046939210033712259959278936226797226705319563152881651856737653 0xy\
   2-7944384420965765593261494180454953086406759682389032896347338575538120 2\
   8961185209y3-724004838368725173328547499687089809774031660296215270015992\
   8043492729323644090 4x4-30229072594889590776353928323077564115059030720231\
   259170516824974832233111085158 0x3y-64037256345646994117946972694997382287\
   2311437350694996140858866680579218287870890x2y2-532379863646322138624707 5\
   735051330232774871673851044154786608529510234932079859 56xy3-1218975423018\
   75587875922379817225860578362871207100484041370955583839312794525742y4+24\
   33708347047682522966825474186845326745189590371631975744516228783111171394\
   6624x5+1606216373645153668593902428268721695221089762034067034776284838852\
   1966758497249202x4y+5651598303207445294562384969689345589163220523450969696\
   72239871636933016905394448563x3y2+960858204298224279213338534542517287703\
   7924552091271096969608386803550323301921788x2y3+8558235582733260502366947\
   804919395067309048344239526133030288583921823321576965835xy4+303725956231\
   5639703927788489158427319647661112642428044888443421378912780120590032y5+\
   6594575731545804603487398646507290337322935660775550288839600875550839097\
   9096832x5y+64726178487018508374852094560329396912914308157744620883518214\
   5224070781590645580x4y2+8062187186325397820006344762655011390551160063929\
   60767130269538721689211948176601x3y3+982993532059947857231143825111343938\
   30917596005312711536952697817470554603613335x2y4-323544315886988640440349\
   4749301588250075322885206313446249770192955204674513532918xy5-29276245067\
   83603552250465706062672211936068566865861780070233558502317790902754400y6\
   -426470069158487929630233924149806719582193092645283568516127585506810587\
   90015680x5y2-473364332920030604571866539342931025537410667628763205622988\
   6027090832623100003518x4y3-1629540748302590439844420378448593093278163922\
   4804004993348846716922569888231976567x3y4-2107016380434273363158526871579\
   9131718541035588982805252185331763057593815591253850x2y5-2712753700556723\
   3548563793286096257634736026320672219442554361516546592978138603437xy6-16\
   6236453093064181321791768662078194711778645862284978284783476400170455038\
   69298816y7-22462580489002698890433637219784330210530427214792451283350598\
   1811605816503156544x5y3-2525993733524834176653909739877411673764944008814\
   873834891914447337016986355178992x4y4+17201007003760048743651399429355760\
   05118968660407248061255704088802708357745470873x3y5+344080412053657256128\
   0999794614866609535365717150295839842851780931017404306291199x2y6-2709680\
   3755172591765452808090668954763775651169235437031274034869589810243240310\
   539xy7-247267989024660062085743410263904820235758607885459118736592152660\
   33012881835530817y8+19896684488050786680787823518626060910842191835123320\
   4681938840749151389529122944x5y4+1268094120866087493081378876883927577098\
   848536145770101493748197034222042058187258x4y5+34081370285041024307277262\
   846402432839759113247786083098221134062815424310614057471x3y6+72310569966\
   133150794172076258509553480300124405678811107513872235037389583475861507x\
   2y7+479146133456814092788747446518150779366321978746645097706516801034958\
   96053211297113xy8+8616286388603703165706430598010290328100760098977244864\
   547132574988860127820428763y9-2199259594746955029413054842313835254351718\
   7654258217105512080427797106068510720x5y5+2032276182023560708144664632281\
   2063635168669994756687724061447964331916607660811908x4y6+1111681674324529\
   44756413433871522598601726444844372668986742835479678142716211588221x3y7+\
   1656835328496245088783015706358167539487399021206218217073417188222513508\
   62946657597x2y8+828135032309779658058556113366038778866240380859960065349\
   62194803728752421265317423xy9+7993173782278607737429474363843914942006054\
   996822806612584783326016637084035325419y10+109962979737347751470652742115\
   69176271758593827129108552756040213898553034255360x4y7-103810785367750869\
   72776123632260497380879214937505247706611808261277642472456775778x3y8-538\
   2455492925062701854397128603717627623134850212032305142753697781246338236\
   4236228x2y9-7680546744094131492663613932021281988489087396947970766859249\
   8046273961982758803427xy10-3299282010894863133641231771817796785660122251\
   8334773418854359891831776392835834337y11+10446248212896863605853285506632\
   2978559118926308347652190620920787182894234786944x3y9+6397723657842261861\
   978522049352610303262270390953012131732278925874034782464421296x2y10-1773\
   6652722812159948092348293452683292142761379510717304587194980969150700013\
   348171xy11-21648004043585490238348105166600872737923911611945206412329816\
   768286605765675723023y12+21041907610451540342015388628008056699134637498З\
   611037558350880045935976114644224x3y10+4474630785433975229581366780744686\
   309294261988261036105549742782501750613281905040x2y11-1219237501662523856\
   43340914660458591334273119150304768107450845197302177103772187267xy12-116\
   5381212725529352830899083069102845544941878031827652305259927665831889091\
   85468983y13+1699246740245260913628159790480763631286813879899641353605366\
   093130456279556824128x3y11-6717085688663586771951129601267760718890209857\
   343282973400977449349317357085492176x2y12-5953208660216712176462904477424\
   23084862149212012151428734823220813694715237150681151xy13-564743825378574\
   869290582813438277493609135692981551278442652814070806661955952981643y14+\
   1433878482657909527566695006980497415544113666641100842367699582875423940\
   6739907584x3y12+299605655305965541701628742774217156664070128731617533054\
   12468414082363775502423344x2y13-14235067204377575006951571088182725384594\
   58268998918535705186603861966378223141538882xy14-133223136222406334831635\
   1273214745985463585116633396174659328304118590750871513503366y15]
↦     _[4]=[0,13931943131582340770912852949915453026787668378663836567494955\
   5000xy+139319431315823407709128529499154530267876683786638365674949555000\
   y2-21303065164905409355910746821643349934000694346271714545358765232x3-12\
   383447741211908993680181433262864291868537072554923863943935148949x2y-247\
   73543381794238558879609013100650724098477681742811225092640516152xy2-1241\
   1398705747234974555338326659429782163941303534159075694064132435y3-830962\
   811772260024920477884851476648380281203104881224150089436944x3y-487411142\
   4037166292027217183353262511573411834969422537249624134592x2y2-4741804097\
   781986980382227974870927301576534280457934293247509748402xy3-698655485517\
   080713275488676369141438383403648593392980147975050754y4+4260613032981081\
   8711821493643286699868001388692543429090717530464x5+248308046779185342154\
   28095106990658633539076228148662871523946593594x4y+8704620416861650389784\
   5557002987836357618225043132600605507566479447x3y2+1484194910694401893580\
   11749187282707122839152159360672672867185366626x2y3+129690875082230637137\
   950680400874694709358932510858238174423875965117xy4+442259867903973453196\
   13857250728224661806313974107238493005775789808y5+11046478982812264190044\
   41651706335175689055671063208985600380653888x5y+9990697317173032726113900\
   802452839134067639644321003181604964934784x4y2+10992048980777511641009030\
   467695016787996751828526502655918650601939x3y3+34349649164520994348527186\
   74724440297203196761895683826623843433879x2y4-535758734608666540298548392\
   74020388382412234229618122839462899517878xy5-5247123566344226397456030797\
   7012486195456757558871224096877918100464y6-654942232454788779379767860352\
   597367763315535271869697890242470128x5y2-75041329977893339040317743680181\
   816767648346639884126272800055730838x4y3-25172776163090560207174166573703\
   5915455315010149425221764835844253281x3y4-3286878042921553313164722228488\
   79867599400341098345065605361508868616x2y5-451625257842369564594876968857\
   655863970925605492620308538466553898751xy6-283967498799420727351156384388\
   905179728164689631586825865867521716956y7-3885760581204356750013501001210\
   537800485242842329279907775404107192x5y3-50985550847662346883652513013374\
   636207226299565077057985110699251072x4y4+11197537473954069016233608966174\
   257402622820220668202993811700156667x3y5+55893720513081539807339034836247\
   563815590447396968650077460312842655x2y6-44286116804176561614294173767931\
   5261579577498907888134555291346676635xy7-40975817533747935979239202790549\
   5121708057780529185890901989678885143y8+320313967369527509295116230370887\
   6949301545552350059765341231800616x5y4+4276446948871629699025913413360173\
   3446463415269456118348670977957610x4y5+6488339452626821470939781957813243\
   22388404747014315457004007505629741x3y6+130991807505346256819574340818231\
   0230916280920024139066689143912165033x2y7+8508146842786427885159089788549\
     43992718583652909741589227723055849403xy8+1501692246728413455168357946240\
     65227693545610182811920959256903157117y9+39881361062399044562540709872293\
     7743188691892992327539959010781616x5y5+3212710065978859005115948578562282\
     67991100718643383365605514894710444x4y6+178528239307575411827078182979564\
     2643866951489787329489947684852740879x3y7+2664786673159464299081021373204\
     820967665477618032556009617347981304943x2y8+13128671107962111960414806946\
     28798927861719700512669052046486656739745xy9+1131872348818182222038173431\
     09611046466620928935984686139642392021853y10-1994068053119952228127035493\
     61468871594345946496163769979505390808x4y7-164194064491321758357747609358\
     226636892595480471921070228562886986384x3y8-87307474644040614422647345474\
     0024461613888740838954751325409404553200x2y9-1281101492267754833486875242\
     467723120825769959117798884375885275683345xy10-56610481457847208304636080\
     1644301134321644930132229206257890755320471y11+17716270649866274103554611\
     26244615457619578516603112269381664499296x3y9+107001388808513925877623349\
     992108088907129198212066490326585619068948x2y10-3646418545145425900645260\
     34018365955136164354894304983284057232670117xy11-433729703139485122588501\
     408950779268449447675287132498425501614757769y12+360410315934694561074114\
     4653259856402227059134963120205061334644400x3y10+693712669211924509432239\
     69818501233031933209624614816341277674789936x2y11-21796077810439590846040\
     88846772531793073092013249389053280019350781109xy12-209450712981380244886\
     4294900410605997058384026396302049186189878056545y13+18190583354114291530\
     298563451821524423499661653123317256140270862832x3y11-1699476738863209814\
     27026767707210461506221842114484304541012959187244x2y12-10361752615643532\
     895528364702773341687030309957197647412580478560294969xy13-98641487846078\
     64553628846890859655440498342258055057496966587459993601y14+2153413383420\
     64203237724900368360311357267361567094896515055823975160x3y12+39561393186\
     8229667642625657783981682236828187985466575178797954569236x2y13-241810069\
     54550610168202557367476669472219588419182001193498265782381544xy14-226448\
     69248012390569132461126480909721726229549315543455587071226705994y15]
↦      _[5]=[0,y15]
↦    [2]:
↦      _[1]=[0,y14]
↦      _[2]=[0,y13+483/16y14]
↦      _[3]=[0,y12+4484/375y13+8614/125y14]
↦      _[4]=[0,y11+11y12+58y13+363/2y14]
↦      _[5]=[0,y10+10y11+48y12+287/2y13+991/4y14]
↦      _[6]=[0,y9+9y10+389/10y11+2193/20y12+1063/5y13+3781/20y14]
↦      _[7]=[0,y8+8y9+154/5y10+84y11+4949/25y12+10188/25y13+16034/25y14]
↦      _[8]=[0,y7+89/2y8+321y9+1140y10+2685y11+10109/2y12+8409y13+14614y14]
↦      _[9]=[0,y6+6y7+33/2y8+63/4y9-357/4y10-1065/2y11-1609y12-13131/4y13-807\
     9/2y14]
↦      _[10]=[0,y5+5y6+23/2y7+57/4y8-27/2y9-139y10-945/2y11-4823/4y12-5173/2y\
     13-4657y14]
↦      _[11]=[0,x+y+20y9+180y10+778y11+2318y12+5752y13+12636y14]
```

D.6.9.4 modVStd

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: modVStd((M, N, I,bound, #); M is a submodule in A^r over the basering=:A, N ist a submodule in R^r over the subring R of the basering generated by the entries of I

Compute: a standard basis of A^r/M+N+maxideal(bound)*A^r using modular version

Return: a list whose ist entry gives a list where each entry is a genrator of standard basis and second entry gives a list of generators after the reduced echelon form

Note: if # is not empty the bound is corrected by computing determinacy

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
module M=maxideal(1)*jacob(I);
module N=I*freemodule(2);
modVStd(M,N,I,15);
↦ [1]:
↦    _[1]=[x+x3+4x2y+3xy2+2x3y+6x2y2+4xy3+5x2y4+6xy5+30x2y5+35xy6+77x2y6+88\
   xy7+136x2y7+153xy8+234x2y8+260xy9+360x2y9+396xy10+385x2y10+420xy11+252x2y\
   11+273xy12+91x2y12+98xy13+14x2y13+15xy14,3x2+4xy+2x2y+3xy2+5xy4+30xy5+77x\
   y6+136xy7+234xy8+360xy9+385xy10+252xy11+91xy12+14xy13]
↦    _[2]=[y+x2y+4xy2+3y3+2x2y2+6xy3+4y4+5xy5+6y6+30xy6+35y7+77xy7+88y8+136\
   xy8+153y9+234xy9+260y10+360xy10+396y11+385xy11+420y12+252xy12+273y13+91xy\
   13+98y14+14xy14+15y15,3xy+4y2+2xy2+3y3+5y5+30y6+77y7+136y8+234y9+360y10+3\
   85y11+252y12+91y13+14y14]
↦    _[3]=[0,x2+20295732609974256678351517467381 74/914290180625267404551036\
   030229087xy+111528308037215826328411571650 9087/91429018062526740455103603\
   0229087y2-30733436108312683243381117471872/914290180625267404551036030229\
   087x3-80128290556229107240472609502872071361548598882425479340633403208 93\
   96045463164633/36200241918436258666427374984354490488701583014810763500 79\
   9640217463646618220452x2y-79725224674267189523469605016856129979639468113\
   398613352659781576440825928368826 5/181001209592181293332136874921772452 44\
   350791507405381750399820108731823309110226xy2-79443844209657655932614941 8\
   045495308640675968238903289634733857553812028961185209/362002419184362586\
   6642737498435449048870158301481076350079964021746364661822045 2y3-2x4-7557\
   268148722397694088482080769391028764757680057814792629206243708058277771 2\
   895/905006047960906466660684374608862262217539575370269087519991005436591\
   1654555113x3y-320186281728234970589734863474986911436155718675347498070 42\
   943334028960914393544 5/181001209592181293332136874921772452443507915074 05\
   381750399820108731823309110226x2y2-133094965911580534656176893376283255 81\
   937179184627610386966521323775587330199648 9/905006047960906466660684374 60\
   886226221753957537026908751999100543659116545551 13xy3-12967823649135700 83\
   786408295927934687003860331990430681291180378551482051005593/385108956579\
   10913474922739345057968605001684058309322873191106614323028317255 8y4+6146\
   687221662536648676223494374 4/914290180625267404551036030229087x5+80310818\
   682257683429695121413436084761054488101703351738814241926098337924862460 1\
   /181001209592181293332136874921772452443507915074053817503998201087318233\
   09110226x4y+565159830320744529456238496968934558916322052345096967223987 1\
   636933016905394448563/362002419184362586664273749843544904887015830148107\
635007996402174636466182204 52x3y2+24021455107455606980333463363562932192 5\
```

```
6350079964021746364661822045 2x3y2+240214551074556069803334633635629321925\
4307277262846402432839759113247786083098221134062815424310614057471/36200\
2419184362586664273749843544904887015830148107635007996402174636466182204\
52x3y6+153852276523687554881217183528743730809149200863146406611731643053\
2710416669699181/77021791315821826949845478690115937210003368116618645746\
3822132286460566345116x2y7+4791461334568140927887474465181507793663219787\
4664509770651680103495896053211297113/36200241918436258666427374984354490\
488701583014810763500799640217463646618220452xy8+861628638860370316570643\
0598010290328100760098977244864547132574988860127820428763/36200241918436\
258666427374984354490488701583014810763500799640217463646618220452y9-5498\
1489868673875735326371057845881358792969135645542763780201069492765171276\
80/9050060479609064666606843746088622622175395753702690875199910054365911\
654555113x5y5+50806904550589017703616615807030159087921674986891719310153\
61991082979151915202977/9050060479609064666606843746088622622175395753702\
690875199910054365911654555113x4y6+11116816743245294475641343387152259860\
1726444844372668986742835479678142716211588221/36200241918436258666427374\
984354490488701583014810763500799640217463646618220452x3y7+16568353284962\
4508878301570635816753948739902120621821707341718822251350862946657597/36\
2002419184362586664273749843544904887015830148107635007996402174636466182\
20452x2y8+828135032309779658058556113366038778866240380859960065349621948\
03728752421265317423/3620024191843625866642737498435449048870158301481076\
3500799640217463646618220452xy9+79931737822786077374294743638439149420060\
54996822806612584783326016637084035325419/3620024191843625866642737498435\
4490488701583014810763500799640217463646618220452y10+27490744934336937867\
66318552892294067939648456782277138189010053474638258563840/9050060479609\
064666606843746088622622175395753702690875199910054365911654555113x4y7-51\
9053926838754348638806181613024869043960746875262385330590413063882123622\
8387889/18100120959218129333213687492177245244350791507405381750399820108\
731823309110226x3y8-13456138732312656754635992821509294069057837125530080\
762856884244453115845591059057/905006047960906466660684374608862262217539\
5753702690875199910054365911654555113x2y9-7680546744094131492663613932021\
2819884890873969479707668592498046273961982758803427/36200241918436258666\
427374984354490488701583014810763500799640217463646618220452xy10-32992820\
1089486313364123177181779678566012225183347734188543598918317763928358343\
37/3620024191843625866642737498435449048870158301481076350079964021746364\
6618220452y11+26115620532242159014633213766580744639779731577086913047655\
230196795723558696736/905006047960906466660684374608862262217539575370269\
0875199910054365911654555113x3y9+1599430914460565465494630512338152575815\
567597738253032933069731468508695616105324/905006047960906466660684374608\
8622622175395753702690875199910054365911654555113x2y10-177366527228121599\
48092348293452683292142761379510717304587194980969150700013348171/3620024\
1918436258666427374984354490488701583014810763500799640217463646618220452\
xy11-21648004043585490238348105166600872737923911611945206412329816768286\
605765675723023/362002419184362586664273749843544904887015830148107635007\
99640217463646618220452y12+5260476902612885085503847157002014174783659374\
5902759389587720011483994028661056/90500604796090646666068437460886226221\
75395753702690875199910054365911654555113x3y10+11186576963584938073953416\
95186171577323565497065259026387435695625437653320476260/9050060479609064\
666606843746088622622175395753702690875199910054365911654555113x2y11-1219\
2375016625238564334091466045859133427311915030476810745084519730217710377\
2187267/36200241918436258666427374984354490488701583014810763500799640217\
463646618220452xy12-11653812127255293528308990830691028455449418780318276\
5230525992766583188909185468983/36200241918436258666427374984354490488701\
583014810763500799640217463646618220452y13+424811685061315228407039947620\
     190907821703469974910338401341523282614069889206032/905006047960906466660\
     6843746088622622175395753702690875199910054365911654555113x3y11-167927142\
     2165896692987782400316940179722552464335820743350244362337329339271373044\
     /905006047960906466660684374608862262217539575370269087519991005436591165\
     4555113x2y12-595320866021671217646290447742423084862149212012151428734823\
     220813694715237150681151/362002419184362586664273749843544904887015830148\
     10763500799640217463646618220452xy13-564743825378574869290582813438277493\
     609135692981551278442652814070806661955952981643/362002419184362586664273\
     74984354490488701583014810763500799640217463646618220452y14+3584696206644\
     773818916737517451243538860284166602752105919248957188559851684976896/905\
     0060479609064666606843746088622622175395753702690875199910054365911654555\
     113x3y12+7490141382649138542540718569355428916601753218290438326353117103\
     520590943875605836/905006047960906466660684374608862262217539575370269087\
     5199910054365911654555113x2y13-711753360218878750347578554409136269229729\
     134499459267852593301930983189111570769441/181001209592181293332136874921\
     77245244350791507405381750399820108731823309110226xy14-666115681112031674\
     158175636607372992731792558316698087329664152059295375435756751683/181001\
     2095921812933321368749217724524435079150740538175039982010873182330911022\
     6y15]
↦    _[4]=[0,xy+y2-269020070934577946/1759358242817180625x3-123834477412119\
     08993680181433262864291868537072554923863943935148949/1393194313158234077\
     09128529499154530267876683786638365674949555000x2y-3096692922724279819859\
     951126637581340512309710217851403136580064519/174149289144779259636410661\
     87394316283484585473329795709368694375xy2-2482279741149446994911067665331\
     885956432788260706831815138812826487/278638862631646815418257058998309060\
     53575336757327673134989911000y3-34623450490510834371686578535478193682511\
     716796036717672920393206/580497630482597532121368872913143876116152849110\
     9931903122898125x3y-60926392800464578650340214791915781394667647937117781\
     7156203016824/17414928914477925963641066187394316283484585473329795709368\
     694375x2y2-23709020488909934901911139874354636507882671402289671466237548\
     74201/69659715657911703854564264749577265133938341893319182837474777500xy\
     3-2477501721691775578991094597053693043912778895721251702652393797/494040\
     536580934069890526700351611809460555616264675055584927500y4+5380401418691\
     55892/1759358242817180625x5+124154023389592671077140475534953293167695381\
     14074331435761973296797/6965971565791170385456426474957726513393834189331\
     9182837474777500x4y+87046204168616503897845557002987836357618225043132600\
     605507566479447/139319431315823407709128529499154530267876683786638365674\
     949555000x3y2+74209745534720094679005874593641353561419576079680336336433\
     592683313/696597156579117038545642647495772651339383418933191828374747775\
     00x2y3+432302916940768790459835601336248982364529775036194127248079586550\
     39/46439810438607802569709509833051510089292227928879455224983185000xy4+5\
     528248348799668164951732156341028082725789246763404811625721973726/174149\
     28914477925963641066187394316283484585473329795709368694375y5+46026995761\
     717767458518402154430632320377319627633707733349193912/580497630482597532\
     1213688729131438761161528491109931903122898125x5y+41627905488220969692141\
     2533435534963919484985180041799233540205616/58049763048259753212136887291\
     31438761161528491109931903122898125x4y2+366401632692583721366967682256500\
     5595998917276175500885306216867313/46439810438607802569709509833051510089\
     292227928879455224983185000x3y3+34349649164520994348527186747244402972031\
     96761895683826623843433879/1393194313158234077091285294991545302678766837\
     86638365674949555000x2y4-569956100647517596062285524191706259387364193932\
     107689781520207637/148212160974280220967158010105483542838166684879402516\
     6754782500xy5-65589044579302829968200384971265607744320946948589030121097\
     39762558/17414928914477925963641066187394316283484585473329795709368694375\
     5y6-27289259685616199140823660848024890323471480636327904078760102922/580\
     4976304825975321213688729131438761161528491109931903122898125x5y2-3752066\
     4988946669520158871840090908383824173319942063136400027865419/69659715657\
     911703854564264749577265133938341893319182837474777500x4y3-83909253876968\
     534023913888579011971818438336716475073921611948084427/464398104386078025\
     69709509833051510089292227928879455224983185000x3y4-136953251788398054715\
     19675952036661149975014212431044400223396202859/5804976304825975321213688\
     729131438761161528491109931903122898125x2y5-45162525784236956459487696885\
     7655863970925605492620308538466553898751/13931943131582340770912852949915\
     4530267876683786638365674949555000xy6-23663958233285060612596365365742098\
     310680390802632235488822293476413/116099526096519506424273774582628775223\
     23056982219863806245796250y7-10334469630862650930886970747900366490652241\
     601939574222806925817/370530402435700552417895025263708857095416712198506\
     291688695625x5y3-21243979519859311201521880422239431753010958152115440827\
     12945802128/5804976304825975321213688729131438761161528491109931903122898\
     125x4y4+11197537473954069016233608966174257402622820220668202993811700156\
     667/139319431315823407709128529499154530267876683786638365674949555000x3y\
     5+11178744102616307961467806967249512763118089479393730015492062568531/27\
     863886263164681541825705899830906053575336757327673134989911000x2y6-29524\
     077869451041076196115845287684105305166593859208970352756445109/928796208\
     7721560513941901966610302017858445585775891044996637000xy7-13658605844582\
     6453264130675968498373902685926843061963633996559628381/46439810438607802\
     569709509833051510089292227928879455224983185000y8+8518988493870412481253\
     091233268289758780706256250158950375616491/370530402435700552417895025263\
     708857095416712198506291688695625x5y4+90988232954715525511189647092769645\
     630773223977566209252491442463/296424321948560441934316020210967085676333\
     369758805033350956500x4y5+64883394526268214709397819578132432238840474701\
     4315457004007505629741/13931943131582340770912852949915453026787668378663\
     8365674949555000x3y6+27870597341563033365866688102515553682800597702179019\
     2908279657705639/29642432194856044193431602021096708567633336975880503335\
     09565000x2y7+850814684278642788515908978854943992718583652909741589227723\
     055849403/139319431315823407709128529499154530267876683786638365674949555\
     000xy8+150169224672841345516835794624065227693545610182811920959256903157\
     117/139319431315823407709128529499154530267876683786638365674949555000y9+\
     49851701327998805703175887340367217898586486624040942494876347702/1741492\
     8914477925963641066187394316283484585473329795709368694375x5y5+2677258388\
     3157158375966238154685688999258393220281947133792907892537/11609952609651\
     950642427377458262877522323056982219863806245796250x4y6+17852823930757541\
     18270781829795642643866951489787329489947684852740879/1393194313158234077\
     09128529499154530267876683786638365674949555000x3y7+266478667315946429908\
     1021373204820967665477618032556009617347981304943/13931943131582340770912\
     8529499154530267876683786638365674949555000x2y8+8752447405308074640276537\
     9641919928524114646700844603469765777115983/92879620877215605139419019666\
     10302017858445585775891044996637000xy9+3772907829393940740127244770320368\
     2155540309645328228713214130673951/46439810438607802569709509833051510089\
     292227928879455224983185000y10-249258506639994028515879436701836089492932\
     43312020471247438173851/1741492891447792596364106618739431628348458547332\
     9795709368694375x4y7-6841419353805073264906150389926109870524811686330044\
     592856786957766/580497630482597532121368872913143876116152849110993190312\
     2898125x3y8-1455124577400676907044122424566707436023147901398257918875682\
     340922/232199052193039012848547549165257550446461139644397276124915925x2y\
     9-256220298453550966697375048493544624165153991823559776875177055136669/2\
     7863886263164681541825705899830906053575336757327673134989911000xy10-5661\
     04814578472083046360801644301134321644930132229206257890755320471/1393194\
     31315823407709128529499154530267876683786638365674949555000y11+7381779437\
     4442808764810880260192310734149104858463011224236020804/58049763048259753\
     21213688729131438761161528491109931903122898125x3y9+267503472021284814694\
     05837498027022226782299553016622581646404767237/3482985782895585192728213\
     2374788632566969170946659591418737388750x2y10-364641854514542590064526034\
     018365955136164354894304983284057232670117/139319431315823407709128529499\
     154530267876683786638365674949555000xy11-43372970313948512258850140895077\
     9268449447675287132498425501614757769/13931943131582340770912852949915453\
     0267876683786638365674949555000y12+18020515796734728053705723266299282011\
     135295674815601025306673222/696597156579117038545642647495772651339383418\
     933191828374747775x3y10+8671408365149056367902996227312654128991651203076\
     852042659709348742/174149289144779259636410661873943162834845854733297957\
     09368694375x2y11-72653592701465302820136294892417726435769733774979635109\
     3339783593703/46439810438607802569709509833051510089292227928879455224983\
     185000xy12-41890142596276048977285898008212119941167680527926040983723797\
     5611309/2786388626316468154182570589983090605357533675732767313498911000\
     y13+2273822919264286441287320431477690552937457706640414657017533857854/1\
     7414928914477925963641066187394316283484585473329795709368694375x3y11-424\
     86918471580245356756691926802615376555460528621076135253239796811/3482985\
     7828955851927282132374788632566969170946659591418737388750x2y12-345391753\
     8547844298509454900924447229010103319065882470860159520098323/46439810438\
     607802569709509833051510089292227928879455224983185000xy13-98641487846078\
     64553628846890859655440498342258055057496966587459993601/1393194313158234\
     07709128529499154530267876683786638365674949555000y14+5383533458551605080\
     943122509209007783931684039177372412876395599379/348298578289558519272821\
     3237478863256696917094665959141873738875x3y12+989034829670574169106564144\
     45995420559207046996366643794699488642309/3482985782895585192728213237478\
     8632566969170946659591418737388750x2y13-302262586931882627102531967093458\
     3684027448552397750149187283222797693/17414928914477925963641066187394316\
     283484585473329795709368694375xy14-37741448746687317615220768544134849536\
     21038258219257242597845204450999/2321990521930390128485475491652575504464\
     6113964439727612491592500y15]
  ↦     _[5]=[0,y15]
  ↦  [2]:
  ↦     _[1]=[0,y14]
  ↦     _[2]=[0,y13+483/16y14]
  ↦     _[3]=[0,y12+4484/375y13+8614/125y14]
  ↦     _[4]=[0,y11+11y12+58y13+363/2y14]
  ↦     _[5]=[0,y10+10y11+48y12+287/2y13+991/4y14]
  ↦     _[6]=[0,y9+9y10+389/10y11+2193/20y12+1063/5y13+3781/20y14]
  ↦     _[7]=[0,y8+8y9+154/5y10+84y11+4949/25y12+10188/25y13+16034/25y14]
  ↦     _[8]=[0,y7+89/2y8+321y9+1140y10+2685y11+10109/2y12+8409y13+14614y14]
  ↦     _[9]=[0,y6+6y7+33/2y8+63/4y9-357/4y10-1065/2y11-1609y12-13131/4y13-807\
     9/2y14]
  ↦     _[10]=[0,y5+5y6+23/2y7+57/4y8-27/2y9-139y10-945/2y11-4823/4y12-5173/2y\
     13-4657y14]
  ↦     _[11]=[0,x+y+20y9+180y10+778y11+2318y12+5752y13+12636y14]
```

D.6.9.5 modVStd0

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: modVStd0((M, N, I,bound, #); M is a submodule in A^r over the basering=:A, N ist a submodule in R^r over the subring R of the basering generated by the entries of I

Compute: a standard basis of M+N+maxideal(bound)*A^r using the parallel modular version

Return: a list whose ist entry gives a list where each entry is a genrator of standard basis and second entry gives a list of generators after the reduced echelon form

Note: if # is not empty the bound is corrected by computing determinacy

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
module M=maxideal(1)*jacob(I);
module N=I*freemodule(2);
modVStd0(M,N,I,15);
↦ [1]:
↦    _[1]=[x+x3+4x2y+3xy2+2x3y+6x2y2+4xy3+5x2y4+6xy5+30x2y5+35xy6+77x2y6+88\
   xy7+136x2y7+153xy8+234x2y8+260xy9+360x2y9+396xy10+385x2y10+420xy11+252x2y\
   11+273xy12+91x2y12+98xy13+14x2y13+15xy14,3x2+4xy+2x2y+3xy2+5xy4+30xy5+77x\
   y6+136xy7+234xy8+360xy9+385xy10+252xy11+91xy12+14xy13]
↦    _[2]=[y+x2y+4xy2+3y3+2x2y2+6xy3+4y4+5xy5+6y6+30xy6+35y7+77xy7+88y8+136\
   xy8+153y9+234xy9+260y10+360xy10+396y11+385xy11+420y12+252xy12+273y13+91xy\
   13+98y14+14xy14+15y15,3xy+4y2+2xy2+3y3+5y5+30y6+77y7+136y8+234y9+360y10+3\
   85y11+252y12+91y13+14y14]
↦    _[3]=[0,x2+2029573260997425667835151746738174/914290180625267404551036\
   030229087xy+1115283080372158263284115716509087/91429018062526740455103603\
   0229087y2-30733436108312683243381117471872/914290180625267404551036030229\
   087x3-8012829055622910724047260950287207136154859888242547934063340320893\
   96045463164633/3620024191843625866642737498435449048870158301481076350079\
   9640217463646618220452x2y-79725224674267189523469605016856129979639468113\
   3986133526597815764408259283688265/18100120959218129333213687492177245244\
   350791507405381750399820108731823309110226xy2-794438442096576559326149418\
   045495308640675968238903289634733857553812028961185209/362002419184362586\
   66427374984354490488701583014810763500799640217463646618220452y3-2x4-7557\
   2681487223976940884820807693910287647576800578147926292062437080582777712\
   895/905006047960906466660684374608862262217539575370269087519991005436591\
   1654555113x3y-32018628172823497058973486347498691143615571867534749807042\
   9433340289609143935445/18100120959218129333213687492177245244350791507405\
   381750399820108731823309110226x2y2-13309496591158053465617689337628325581\
   9371791846276103869665213237755873301996489/90500604796090646666068437460\
   886226221753957537026908751999100543659116545551113xy3-1296782364913570083\
   7864082959279346870038603319904306812911803785514820510055593/385108956579\
   1091347492273934505796860500168405830932287319110661432302831725558y4+6146\
   6872216625366486762234943744/9142901806252674045510360302290887x5+80310818\
   6822576834296951214134360847610544881017033517388142419260983379248624601\
   /181001209592181293332136874921772452443507915074053817503998201087318233\
   09110226x4y+565159830320744529456238496968934558916322052345096967223987 1\
   636933016905394448563/362002419184362586664273749843544904887015830148107\
   63500799640217463646618220452x3y2+2402145510745560698033346336356293219 25\
   9481138022817774242402096700887580825480447/905006047960906466660684374 60\
   886226221753957537026908751999100543659116545555113x2y3+855823558273326050\
   2366947804919395067309048344239526133030288583921823321576965835/36200241\
   918436258666427374984354490488701583014810763500799640217463646618220452x\
   y4+759314890578909925981947122289606829911915278160607011222110855344728 1\
   95030147508/905006047960906466660684374608862262217539575370269087519991 0\
   054365911654555113y5+164864393288645115087184966162682258433073391519388 7\
   5722099002188877097744774208/90500604796090646666068437460886226221753957\
   537026908751999100543659116545555113x5y+16181544621754627093713023640082 34\
   922822857703943615522087955363060176953976613 95/90500604796090646666068 43\
   74608862262217539575370269087519991005436591165455 5113x4y2+80621871863253\
   97820006344762655011390551160063929607671302695387216892119481766 01/36200\
   241918436258666427374984354490488701583014810763500799640217463646618220 4\
   52x3y3+982993532059947857231143825111343938309175960053127115369526978174\
   70554603613335/3620024191843625866642737498435449048870158301481076350079\
   964021746364661822 0452x2y4-34419608073083897919186114354272215426333222 18\
   304588772606138503143834760120779 7/38510895657910913474922739345057968605\
   00168405830932287319110661432302831725 58xy5-7319061266959008880626164265 1\
   56680529840171417164654450175583896255794477256886 00/905006047960906466 66\
   06843746088622622175395753702690875199910054365911654555113y6-10661751728\
   962198240755848103745167989554827316132089212903189637670264697503920/905\
   006047960906466660684374608862262217539575370269087519991005436591165455 5\
   113x5y2-236682166460015302285933269671465512768705333814381602811494301 35\
   454163115500017 59/18100120959218129333213687492177245244350791507405381 75\
   039982010873182330911 0226x4y3-162954074830259043984442037844859309327816 3\
   922480400499334884671692256988823197 6567/362002419184362586664273749843 54\
   4904887015830148107635007996402174636466182204 52x3y4-1053508190217136681 5\
   792634357899565859270517794491402626092665881528796907795626925/181001209\
   59218129333213687492177245244350791507405381750399820108731823309110226x2\
   y5-2712753700556723354856379328609625763473602632067221944255436151654659\
   2978138603437/362002419184362586664273749843544904887015830148107635007 99\
   64021746364661822 0452xy6-41559113273266045330447942165519548677944661465 5\
   7124457119586910004261375967324704/9050060479609064666606843746088622622 1\
   7539575370269087519991005436591165455 5113y7-1194818111117164834597533894 6\
   6937926651757591568044953634843607346598838565508 8/1925544782895545673746\
   136967252898430250084202915466143659555330716151415862 79x5y3-631498433381\
   2085441634774349693529184412360022037184587229786118342542465887947 48/905\
   0060479609064666606843746088622622175395753702690875199910054365911654555\
   113x4y4+172010070037600487436513994293557600511896866040724806125570408 88\
   02708357745470873/3620024191843625866642737498435449048870158301481076350\
   0799640217463646618220452x3y5+344080412053657256128099979461486660953536 5\
   71715029583984285178093101740430629119 9/362002419184362586664273749843544\
   904887015830148107635007996402174636466182204 52x2y6-2709680375517259176 54\
   5280809066895476377565116923543703127403486958981024324031053 9/3620024191\
   8436258666427374984354490488701583014810763500799640217463646618220452xy7\
   -247267989024660062085743410263904820235758607885459118736592152660330128\
   81835530817/3620024191843625866642737498435449048870158301481076350079964\
   0217463646618220452y8+105833428127929716387169274035245004844905275718741\
   0663201802344410379731537888/19255447828955456737461369672528984302500842\
   0291546614365955533071615141586279x5y4+1349036298793710099022743486046731\
   4649987750384529469164827108479087468532533907/38510895657910913474922739\
   3450579686050016840583093228731911066143230283172558x4y5+3408137028504102\
   4307277262846402432839759113247786083098221134062815424310614057471/36200\
   2419184362586664273749843544904887015830148107635007996402174636466182204\
   52x3y6+153852276523687554881217183528743730809149200863146406611731643053\
   2710416669699181/77021791315821826949845478690115937210003368116618645746\
   3822132286460566345116x2y7+4791461334568140927887474465181507793663219787\
   4664509770651680103495896053211297113/36200241918436258666427374984354490\
   488701583014810763500799640217463646618220452xy8+861628638860370316570643\
   0598010290328100760098977244864547132574988860127820428763/36200241918436\
   258666427374984354490488701583014810763500799640217463646618220452y9-5498\
   1489868673875735326371057845881358792969135645542763780201069492765171276\
   80/9050060479609064666606843746088622622175395753702690875199910054365911\
   654555113x5y5+50806904550589017703616615807030159087921674986891719310153\
   61991082979151915202977/9050060479609064666606843746088622622175395753702\
   690875199910054365911654555113x4y6+11116816743245294475641343387152259860\
   1726444844372668986742835479678142716211588221/36200241918436258666427374\
   984354490488701583014810763500799640217463646618220452x3y7+16568353284962\
   4508878301570635816753948739902120621821707341718822251350862946657597/36\
   2002419184362586664273749843544904887015830148107635007996402174636466182\
   20452x2y8+828135032309779658058556113366038778866240380859960065349621948\
   03728752421265317423/3620024191843625866642737498435449048870158301481076\
   3500799640217463646618220452xy9+79931737822786077374294743638439149420060\
   54996822806612584783326016637084035325419/3620024191843625866642737498435\
   4490488701583014810763500799640217463646618220452y10+27490744934336937867\
   66318552892294067939648456782277138189010053474638258563840/9050060479609\
   064666606843746088622622175395753702690875199910054365911654555113x4y7-51\
   9053926838754348638806181613024869043960746875262385330590413063882123622\
   8387889/18100120959218129333213687492177245244350791507405381750399820108\
   731823309110226x3y8-13456138732312656754635992821509294069057837125530080\
   762856884244453115845591059057/905006047960906466660684374608862262217539\
   5753702690875199910054365911654555113x2y9-7680546744094131492663613932021\
   2819884890873969479707668592498046273961982758803427/36200241918436258666\
   427374984354490488701583014810763500799640217463646618220452xy10-32992820\
   1089486313364123177181779678566012225183347734188543598918317763928358343\
   37/3620024191843625866642737498435449048870158301481076350079964021746364\
   6618220452y11+26115620532242159014633213766580744639779731577086913047655\
   230196795723558696736/905006047960906466660684374608862262217539575370269\
   0875199910054365911654555113x3y9+1599430914460565465494630512338152575815\
   567597738253032933069731468508695616105324/905006047960906466660684374608\
   8622622175395753702690875199910054365911654555113x2y10-177366527228121599\
   48092348293452683292142761379510717304587194980969150700013348171/3620024\
   1918436258666427374984354490488701583014810763500799640217463646618220452\
   xy11-21648004043585490238348105166600872737923911611945206412329816768286\
   605765675723023/362002419184362586664273749843544904887015830148107635007\
   99640217463646618220452y12+5260476902612885085503847157002014174783659374\
   5902759389587720011483994028661056/90500604796090646666068437460886226221\
      7539575370269087519991005436591165455511 3x3y10+1118657696358493807395341 6\
```

```
      753957537026908751999100543659116545551 13x3y10+11186576963584938073953416\
      951861715773235654970652590263874356956254376533204 76260/9050060479609064\
      66660684374608862262217539575370269087519991005436591165455511 3x2y11-1219\
      2375016625238564334091466045859133427311915030476810745084519730217710377\
      2187267/36200241918436258666427374984354490488701583014810763500799640217\
      463646618220452xy12-1165381212725529352830899083069102845544941878031827 6\
      5230525992766583188909185468983/36200241918436258666427374984354490488701\
      583014810763500799640217463646618220452y13+4248116850613152284070399476 20\
      1909078217034699749103384013415232826140698892060 32/9050060479609064666 60\
      68437460886226221753957537026908751999100543659116545551 13x3y11-167927142\
      21658966929877824003169401797225524643358207433502443623373293392713730 44\
      /90500604796090646666068437460886226221753957537026908751999100543659116 5\
      4555113x2y12-5953208660216712176462904477424230848621492120121514287348 23\
      22081369471523715068115 1/3620024191843625866642737498435449048870158301 48\
      10763500799640217463646618220452xy13-564743825378574869290582813438277493\
      60913569298155127844265281407080666195595298164 3/3620024191843625866642 73\
      7498435449048870158301481076350079964021746364661822045 2y14+3584696206644\
      773818916737517451243538860284166602752105919248957188559851684976896/905\
      00604796090646666068437460886226221753957537026908751999100543659116545 55\
      113x3y12+7490141382649138542540718569355428916601753218290438326353117103\
      520590943875605836/9050060479609064666606843746088622621753957537026908 7\
      5199910054365911654555113x2y13-7117533602188787503475785544091362692297 29\
      1344994592678525933019309831891115707694 41/1810012095921812933321368749 21\
      7724524435079150740538175039982010873182330911022 6xy14-6661156811120316 74\
      1581756366073729927317925583166980873296641520592953754357567516 83/181001\
      20959218129333213687492177245244350791507405381750399820108731823309110 22\
      6y15]
↦    _[4]=[0,xy+y2-26902007093457794 6/1759358242817180625x3-123834477412119\
      25334355349639194849851800417992335402056 16/58049763048259753212136887291\
      31438761161528491109931903122898125x4y2+3664016326925837213669676822565 00\
      5595998917276175500885306216867313/4643981043860780256970950983305151 0089\
      2922279288794552249831850 00x3y3+343496491645209943485271867472444029 72031\
      967618956838266238434338 79/13931943131582340770912852949915453026787 66837\
      866383656749495550 00x2y4-56995610064751759606228552419170625938736 4193932\
      1076897815202076 37/148212160974280220967158010105483542838166684879402516\
      6754782500xy5-655890445793028299682003849712656077443209469485890301210 97\
      39762558/174149289144779259636410661873943162834845854733297957093686943 7\
      5y6-272892596856161991408236608480248903234714806363279040787601029 22/580\
      497630482597532121368872913143876116152849110993190312289 8125x5y2-3752066\
      498894666952015887184009090838382417331994206313640002786541 9/69659715657\
      911703854564264749577265133938341893319182837474777500x4y3-83909253876968\
      5340239138885790119718184383367164750739216119480844 27/464398104386078025\
      69709509833051510089292227928879455224983185000x3y4-1369532517883980547 15\
      19675952036661149975014212431044400223396202859/5804976304825975321213688\
      72913143876116152849110993190312289812 5x2y5-451625257842369564594876968 85\
      7655863970925605492620308538466553898751/1393194313158234077091285294991 5\
      453026787668378663836567494955500 0xy6-236639582332850606125963653657420 98\
      310680390802632235488822293476413/116099526096519506424273774582628775223\
      2305698221986380624579625 0y7-10334469630862650930886970747900366490652241\
      6019395742228069258 17/370530402435700552417895025263708857095416712198506\
      291688695625x5y3-21243979519859311201521880422239431753010958152115440827\
      12945802128/58049763048259753212136887291314387611615284911099319031228 98\
      125x4y4+111975374739540690162336089661742574026228202206682029938117001 56\
      667/13931943131582340770912852949915453026787668378663836567494955500 0x3y\
      5+111787441026163079614678069672495127631180894793937300154920625685 31/27\
      863886263164681541825705899830906053575336757327673134989911000x2y6-29524\
      0778694510410761961158452876841053051665938592089703527564451 09/928796208\
      77215605139419019666103020178584455857758910449966370 00xy7-13658605844582\
      645326413067596849837390268592684306196363399655962838 1/4643981043860780 2\
      56970950983305151008929222792887945522498318500 0y8+851898849387041248125 3\
      09123326828975878070625625015895037561649 1/37053040243570055241789502526 3\
      70885709541671219850629168869562 5x5y4+909882329547155255111896470927696 45\
      6307732239775662092524914424 63/296424321948560441934316020210967085676333\
      36975880503335095650 0x4y5+648833945262682147093978195781324322388404747 01\
      4315457004007505629741/139319431315823407709128529499154530267876683786 63\
      8365674949555000x3y6+27870597341563033365866881025155536828005977021790 19\
      2908279657705639/296424321948560441934316020210967085676333369758805033 35\
      09565000x2y7+85081468427864278851590897885494399271858365290974158922772 3\
      055849403/13931943131582340770912852949915453026787668378663836567494955 5\
      000xy8+150169224672841345516835794624065227693545610182811920959256903157\
      117/139319431315823407709128529499154530267876683786638365674949555000y9+\
      49851701327998805703175887340367217898586486624040942494876347702/1741492\
      891447792596364106618739431628348458547332979570936869437 5x5y5+2677258388\
      3157158375966238154685688999258393220281947133792907892537/11609952609651\
      95064242737745826287752232305698221986380624579625 0x4y6+17852823930757541\
      1827078182979564264386695148978732948994768485274087 9/139319431315823407 7\
      0912852949915453026787668378663836567494955500 0x3y7+2664786673159464299 08\
      1021373204820967665477618032556009617347981304943/139319431315823407709 12\
      852949915453026787668378663836567494955500 0x2y8+87524474053080746402765 37\
      964191992852411464670084460346976577711598 3/928796208772156051394190196 66\
      10302017858445585775891044996637000xy9+37729078293939407401272447703203 68\
   2155540309645328228713214130673951/4643981043860780256970950983305151 0089\
```

```
   2155540309645328228713214130673951/46439810438607802569709509833051510089\
   9/2y14]
↦    _[10]=[0,y5+5y6+23/2y7+57/4y8-27/2y9-139y10-945/2y11-4823/4y12-5173/2y\
   13-4657y14]
↦    _[11]=[0,x+y+20y9+180y10+778y11+2318y12+5752y13+12636y14]
```

D.6.9.6 classifySimpleMaps

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: classifySimpleMaps(I); I=an ideal with 2 generators in a polynomial ring with 2 variables and local ordering defining a map germ C^2 to C^2

Compute: The normal form of the germ in Riegers classification if it is simple

Return: normal form of I, of type ideal

Note: If I is not simple it returns (0,0)

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
classifySimpleMaps(I);
↦ _[1]=x
↦ _[2]=xy+y5+y7
```

D.6.9.7 classifySimpleMaps1

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: classifySimpleMaps1(I); I=an ideal with 2 generators in a polynomial ring with 2 variables and local ordering defining a map germ C^2 to C^2

Compute: The normal form of the germ in Riegers classification if it is simple

Return: normal form of I, of type ideal

Note: If I is not simple it returns (0,0)

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y5+y7;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
```

```
f2=phi(f22);
ideal I=f1,f2;
classifySimpleMaps1(I);
↦ _[1]=x
↦ _[2]=xy+y5+y7
```

D.6.9.8 classifyUnimodalMaps

Procedure from library `classifyMapGerms.lib` (see Section D.6.9 [classifyMapGerms_lib], page 1662).

Usage: classifyUnimodalMaps(I); I an ideal with 2 generators in a polynomial ring with 2 variables and local ordering defining a map germ C^2 to C^2

Compute: The normal form of the germ in Riegers classification if it is simple

Return: normal form of I, of type ideal

Note: If I is not unimodal it returns (0,0)

Example:

```
LIB "classifyMapGerms.lib";
ring R=0,(x,y),(c,ds);
poly f1=x;
poly f2=xy+y6+y9;
poly f11=f1+f2*f1;
poly f22=f2+f1^2;
map phi=basering,x+y,y+y2;
f1=phi(f11);
f2=phi(f22);
ideal I=f1,f2;
classifyUnimodalMaps(I);
↦ _[1]=x
↦ _[2]=xy+y6+y9
```

D.6.10 curvepar_lib

Library: curvepar.lib

Purpose: Resolution of space curve singularities, semi-group

Author: Gerhard Pfister email: pfister@mathematik.uni-kl.de Nil Sahin email: e150916@metu.edu.tr
Maryna Viazovska email: viazovsk@mathematik.uni-kl.de

Procedures: See also: Section D.6.21 [spcurve_lib], page 1767.

D.6.10.1 BlowingUp

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: BlowingUp(f,I,l);
f=poly
b=ideal
l=list

Assume: The basering is r=0,(x(1..n),a),dp
f is an irrreducible polynomial in k[a],
I is an ideal of a curve(if we consider a as a parameter)

Compute: Blowing-up of the curve at point 0.

Return: list C of charts.
Each chart C[i] is a list of size 5 (reps. 6 in case of plane curves) C[i][1] is an integer j. It shows, which standard chart do we consider. C[i][2] is an irreducible poly g in k[a]. It is a minimal polynomial for the new parameter.
C[i][3] is an ideal H in k[a].
c_i=F_i(a_new) for i=1..n,
a_old=H[n+1](a_new).
C[i][4] is a map teta:k[x(1)..x(n),a]–>k[x(1)..x(n),a] from the new curve to the old one.
x(1)–>x(j)*x(1) . . . x(j)–>x(j) . . . x(n)–>x(j)*(c_n+x(n))
C[i][5] is an ideal J of a new curve. J=teta(I).
C[i][6] is the list of exceptional divisors in the chart

Example:

```
LIB "curvepar.lib";
ring r=0,(x(1..3),a),dp;
poly f=a2+1;
ideal i=x(1)^2+a*x(2)^3,x(3)^2-x(2);
list l=1,3,2;
list B=BlowingUp(f,i,l);
B;
↦ [1]:
↦    [1]:
↦       3
↦    [2]:
↦       a^2+1
↦    [3]:
↦       _[1]=0
↦       _[2]=0
↦       _[3]=1
↦    [4]:
↦       _[1]=x(1)*x(3)
↦       _[2]=x(2)*x(3)
↦       _[3]=x(3)
↦       _[4]=a
↦    [5]:
↦       _[1]=x(2)-x(3)
↦       _[2]=x(2)^3*x(3)*a+x(1)^2
```

D.6.10.2 CurveRes

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: CurveRes(I);
I ideal

Assume: The basering is r=0,(x(1..n))
V(I) is a curve with a singular point 0.

Compute: Resolution of the curve V(I).

Return: a ring R=basering+k[a]
Ring R contains a list Resolve
Resolve is a list of charts
Each Resolve[i] is a list of size 6
Resolve[i][1] is an ideal J of a new curve. J=teta(I). Resolve[i][2] ideal which represents the map
teta:k[x(1)..x(n),a]–>k[x(1)..x(n),a] from the
new curve to the old one.
Resolve[i][3] is an irreducible poly g in k[a]. It is a minimal polynomial for the new parameter a. deg(g) gives the number of branches in Resolve[i] Resolve[i][4] sequence of multiplicities (sum over all branches in Resolve as long as they intersect each other !)
Resolve[i][5] is a list of integers l. It shows, which standard charts we considered. Resolve[i][6] HN matrix
Resolve[i][7] (only for plane curves) the development of exceptional divisors the entries correspond to the i-th blowing up. The first entry is an intvec. The first negative entry gives the splitting of the (over Q irreducible) branches. The second entry is a list of the exceptional divisors. If the entry is an integer i, it says that the divisor is not visible in this chart after the i-th blowing up.

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y,z),dp;
ideal i=x2-y3,z2-y5;
def s=CurveRes(i);
setring s;
Resolve;
↦ [1]:
↦    [1]:
↦       _[1]=x(1)
↦       _[2]=x(1)*x(2)-x(3)^2+2*x(3)
↦    [2]:
↦       _[1]=x(1)^2*x(2)^5+2*x(1)*x(2)^4+x(2)^3
↦       _[2]=x(1)*x(2)^3+x(2)^2
↦       _[3]=x(1)^2*x(2)^7*x(3)-x(1)^2*x(2)^7+2*x(1)*x(2)^6*x(3)-2*x(1)*x(2\
   )^6+x(2)^5*x(3)-x(2)^5
↦       _[4]=a
↦    [3]:
↦       a
↦    [4]:
↦       [1]:
↦          4
↦       [2]:
↦          2
↦       [3]:
↦          2
↦       [4]:
↦          2
↦    [5]:
↦       [1]:
↦          2
↦       [2]:
```

```
↦          1
↦       [3]:
↦          2
↦       [4]:
↦          2
↦    [6]:
↦       [1]:
↦          _[1]=0
↦          _[2]=1
↦          _[3]=0
↦       [2]:
↦          _[1]=1
↦          _[2]=0
↦          _[3]=0
↦       [3]:
↦          _[1]=1
↦          _[2]=1
↦          _[3]=0
↦       [4]:
↦          _[1]=0
↦          _[2]=1
↦          _[3]=-1
↦ [2]:
↦    [1]:
↦       _[1]=x(1)
↦       _[2]=x(1)*x(2)-x(3)^2-2*x(3)
↦    [2]:
↦       _[1]=x(1)^2*x(2)^5+2*x(1)*x(2)^4+x(2)^3
↦       _[2]=x(1)*x(2)^3+x(2)^2
↦       _[3]=x(1)^2*x(2)^7*x(3)+x(1)^2*x(2)^7+2*x(1)*x(2)^6*x(3)+2*x(1)*x(2\
  )^6+x(2)^5*x(3)+x(2)^5
↦       _[4]=a
↦    [3]:
↦       a
↦    [4]:
↦       [1]:
↦          4
↦       [2]:
↦          2
↦       [3]:
↦          2
↦       [4]:
↦          2
↦    [5]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦       [3]:
↦          2
↦       [4]:
↦          2
↦    [6]:
```

```
↦       [1]:
↦          _[1]=0
↦          _[2]=1
↦          _[3]=0
↦       [2]:
↦          _[1]=1
↦          _[2]=0
↦          _[3]=0
↦       [3]:
↦          _[1]=1
↦          _[2]=1
↦          _[3]=0
↦       [4]:
↦          _[1]=0
↦          _[2]=1
↦          _[3]=1
```

D.6.10.3 CurveParam

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: CurveParam(I);
I ideal

Assume: I is an ideal of a curve C with a singular point 0.

Compute: Parametrization for algebraic branches of the curve C.

Return: list L of size 1.
L[1] is a ring ring rt=0,(t,a),ds;
Ring R contains a list Param
Param is a list of algebraic branches
Each Param[i] is a list of size 3
Param[i][1] is a list of polynomials
Param[i][2] is an irredusible polynomial f\in k[a].It is a minimal polynomial for the parameter a.
Param[i][3] is an integer b–upper bound for the conductor of Weierstrass semigroup

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y,z),dp;
ideal i=x2-y3,z2-y5;
def s=CurveParam(i);
setring s;
Param;
↦ [1]:
↦    [1]:
↦       [1]:
↦          t3
↦       [2]:
↦          t2
↦       [3]:
↦          -t5
↦    [2]:
↦       a
↦    [3]:
↦       38
↦ [2]:
↦    [1]:
↦       [1]:
↦          t3
↦       [2]:
↦          t2
↦       [3]:
↦          t5
↦    [2]:
↦       a
↦    [3]:
↦       38
```

D.6.10.4 WSemigroup

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: WSemigroup(X,b0);
X a list of polynomials in one variable, say t.
b0 an integer

Compute: Weierstrass semigroup of space curve C,which is given by a parametrization X[1](t),...,X[k](t), till the bound b0.

Assume: b0 is greater then conductor

Return: list M of size 5.
M[1]= list of integers, which are minimal generators set of the Weierstrass semigroup.
M[2]=integer, conductor of the Weierstrass semigroup. M[3]=intvec, all elements of the Weierstrass semigroup till some bound b, which is greater than conductor.

Warning: works only over the ring with one variable with ordering ds

Example:

```
LIB "curvepar.lib";
ring r=0,(t),ds;
list X=t4,t5+t11,t9+2*t7;
list L=WSemigroup(X,30);
L;
↦ [1]:
↦    4,5,7
↦ [2]:
↦    7
↦ [3]:
↦    4,5,7,8,9,10
```

D.6.10.5 primparam

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: MultiplicitySequence(x,y,c); x poly, y poly, c integer

Assume: x and y are polynomials in k(a)[t] such that (x,y) is a primitive parametrization of a plane curve branch and ord(x)<ord(y).

Return: Hamburger-Noether Matrix of the curve branch given parametrically by (x,y).

Example:

```
LIB "curvepar.lib";
ring r=(0,a),t,ds;
poly x=t6;
poly y=t8+t11;
int c=15;
primparam(x,y,c);
↦ _[1,1]=0
↦ _[1,2]=t
↦ _[1,3]=0
↦ _[1,4]=0
↦ _[1,5]=0
↦ _[1,6]=0
↦ _[1,7]=0
↦ _[1,8]=0
↦ _[1,9]=0
↦ _[1,10]=0
↦ _[1,11]=0
↦ _[1,12]=0
↦ _[1,13]=0
↦ _[1,14]=0
↦ _[1,15]=0
↦ _[2,1]=0
↦ _[2,2]=0
↦ _[2,3]=1
↦ _[2,4]=0
↦ _[2,5]=t
↦ _[2,6]=0
↦ _[2,7]=0
↦ _[2,8]=0
↦ _[2,9]=0
↦ _[2,10]=0
↦ _[2,11]=0
↦ _[2,12]=0
↦ _[2,13]=0
↦ _[2,14]=0
↦ _[2,15]=0
↦ _[3,1]=0
↦ _[3,2]=1/9
↦ _[3,3]=0
↦ _[3,4]=0
↦ _[3,5]=-7/243
↦ _[3,6]=0
↦ _[3,7]=0
↦ _[3,8]=250/19683
↦ _[3,9]=0
↦ _[3,10]=0
↦ _[3,11]=-3625/531441
↦ _[3,12]=0
↦ _[3,13]=0
↦ _[3,14]=58351/14348907
↦ _[3,15]=0
```

D.6.10.6 MultiplicitySequence

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: MultiplicitySequence(i); i ideal

Assume: i is the defining ideal of a (reducible) plane curve singularity.

Return: list X of charts. Each chart contains the multiplicity sequence of the corresponding branch.

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y),ds;
ideal i=x14-x4y7-y11;
MultiplicitySequence(i);
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    1
↦ [5]:
↦    7,3,3,1,1,1
```

D.6.10.7 CharacteristicExponents

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: CharacteristicExponents(i); i ideal

Assume: i is the defining ideal of a (reducible) plane curve singularity.

Return: list X of charts. Each chart contains the characteristic exponents of the corresponding branch.

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y),ds;
ideal i=x14-x4y7-y11;
CharacteristicExponents(i);
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    1
↦ [5]:
↦    7,10
```

D.6.10.8 IntersectionMatrix

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: IntersectionMatrix(i); i ideal

Assume: i is the defining ideal of a (reducible) plane curve singularity.

Return: intmat of the intersection multiplicities of the branches.

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y),ds;
ideal i=x14-x4y7-y11;
IntersectionMatrix(i);
↦ 0,1,1,1,7,
↦ 0,0,1,1,7,
↦ 0,0,0,1,7,
↦ 0,0,0,0,7,
↦ 0,0,0,0,0
```

D.6.10.9 ContactMatrix

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: ContactMatrix(I); I ideal

Assume: i is the defining ideal of a (reducible) plane curve singularity.

Return: intmat N of the contact matrix of the braches of the curve.

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y),ds;
ideal i=x14-x4y7-y11;
ContactMatrix(i);
↦ 0,1,1,1,1,
↦ 1,0,1,1,1,
↦ 1,1,0,1,1,
↦ 1,1,1,0,1,
↦ 1,1,1,1,0
```

D.6.10.10 plainInvariants

Procedure from library `curvepar.lib` (see Section D.6.10 [curvepar_lib], page 1681).

Usage: plainInvariants(i); i ideal

Assume: i is the defining ideal of a (reducible) plane curve singularity.

Return: list L of charts. L[j] is the invariants of the jth branch and the last entry of L is a list containing the intersection matrix,contact matrix,resolution graph of the curve.
L[j][1]: intvec (characteristic exponents of the jth branch) L[j][2]: intvec (generators of the semigroup of the jth branch) L[j][3]: intvec (first components of the puiseux pairs of the jth branch) L[j][4]: intvec (second components of the puiseux pairs of the jth branch) L[j][5]: int (degree of conductor of the jth branch) L[j][6]: intvec (multiplicity sequence of the jth branch. L[last][1]: intmat (intersection matrix of the branches) L[last][2]: intmat (contact matrix of the branches)
L[last][3]: intmat (resolution graph of the curve)

Example:

```
LIB "curvepar.lib";
ring r=0,(x,y),ds;
ideal i=x14-x4y7-y11;
plainInvariants(i);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       0
↦    [4]:
↦       0
↦    [5]:
↦       0
↦    [6]:
↦       1
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       0
↦    [4]:
↦       0
↦    [5]:
↦       0
↦    [6]:
↦       1
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       0
↦    [4]:
↦       0
↦    [5]:
↦       0
↦    [6]:
↦       1
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       0
↦    [4]:
↦       0
↦    [5]:
↦       0
↦    [6]:
↦       1
↦ [5]:
↦    [1]:
↦       7,10
↦    [2]:
↦       7,10
↦    [3]:
↦       10
↦    [4]:
↦       7
↦    [5]:
↦       54
↦    [6]:
↦       7,3,3,1,1,1
↦ [6]:
↦    [1]:
↦       0,1,1,1,7,
↦       0,0,1,1,7,
↦       0,0,0,1,7,
↦       0,0,0,0,7,
↦       0,0,0,0,0
↦    [2]:
↦       0,1,1,1,1,
↦       1,0,1,1,1,
↦       1,1,0,1,1,
↦       1,1,1,0,1,
↦       1,1,1,1,0
↦    [3]:
↦       1,1,1,1,1,0,0,1,0,0,0,
↦       1,-1,0,0,0,0,0,0,0,0,0,
↦       1,0,-2,0,0,0,0,0,0,0,0,
↦       1,0,0,-3,0,0,0,0,0,0,0,
↦       1,0,0,0,-4,0,0,0,0,0,0,
↦       0,0,0,0,0,2,1,0,0,0,0,
↦       0,0,0,0,0,1,3,0,0,1,0,
↦       1,0,0,0,0,0,0,4,1,0,0,
↦       0,0,0,0,0,0,0,1,5,1,0,
↦       0,0,0,0,0,0,1,0,1,6,1,
↦       0,0,0,0,0,0,0,0,0,1,-5
```

See also: Section D.6.10.7 [CharacteristicExponents], page 1688; Section D.6.10.9 [ContactMatrix], page 1689; Section D.6.10.8 [IntersectionMatrix], page 1689; Section D.6.10.6 [MultiplicitySequence], page 1688.

D.6.11 deform_lib

Library: deform.lib

Purpose: Miniversal Deformation of Singularities and Modules

Author: Bernd Martin, email: martin@math.tu-cottbus.de

Procedures:

D.6.11.1 versal

Procedure from library `deform.lib` (see Section D.6.11 [deform_lib], page 1691).

Usage: versal(Fo[,d,any]); Fo=ideal, d=int, any=list

Compute: miniversal deformation of Fo up to degree d (default d=100),

Return: list L of 4 rings:
L[1] extending the basering Po by new variables given by "A,B,.." (deformation parameters); the new variables precede the old ones, the ordering is the product of "ls" and "ord(Po)"
L[2] = L[1]/Fo extending Qo=Po/Fo,
L[3] = the embedding ring of the versal base space,
L[4] = L[1]/Js extending L[3]/Js.
In the ring L[1] the following matrices are stored:
Js = giving the versal base space (obstructions),
Fs = giving the versal family of Fo,
Rs = giving the lifting of Ro=syz(Fo).

If d is defined (!=0), it computes up to degree d.
If 'any' is defined and any[1] is no string, interactive version.
Otherwise 'any' is interpreted as a list of predefined strings: "my","param","order","out":
("my" internal prefix, "param" is a letter (e.g. "A") for the name of the first parameter or (e.g. "A(") for index parameter variables, "order" ordering string for ring extension), "out" name of output file).

Note: printlevel < 0 no additional output,
printlevel >=0,1,2,.. informs you, what is going on;
this proc uses 'execute'.

Example:

```
LIB "deform.lib";
int p          = printlevel;
printlevel     = 0;
ring r1        = 0,(x,y,z,u,v),ds;
matrix m[2][4] = x,y,z,u,y,z,u,v;
ideal Fo       = minor(m,2);
// cone over rational normal curve of degree 4
list L=versal(Fo);
↦ // ready: T_1 and T_2
↦ // start computation in degree 2.
↦ 
↦ 
↦ // 'versal' returned a list, say L, of four rings. In L[1] are stored:
↦ //   as matrix Fs: Equations of total space of the miniversal deformation\
   ,
↦ //   as matrix Js: Equations of miniversal base space,
↦ //   as matrix Rs: syzygies of Fs mod Js.
↦ // To access these data, type
↦      def Px=L[1]; setring Px; print(Fs); print(Js); print(Rs);
↦ 
↦ // L[2] = L[1]/Fo extending Qo=Po/Fo,
↦ // L[3] = the embedding ring of the versal base space,
↦ // L[4] = L[1]/Js extending L[3]/Js.
↦
L;
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering ds
↦ //                  : names    A B C D
↦ //        block   2 : ordering ds
↦ //                  : names    x y z u v
↦ //        block   3 : ordering C
↦ [2]:
↦    // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering ds
↦ //                  : names    A B C D
↦ //        block   2 : ordering ds
↦ //                  : names    x y z u v
↦ //        block   3 : ordering C
↦ // quotient ring from ideal ...
↦ [3]:
↦    // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering ds
↦ //                  : names    A B C D
↦ //        block   2 : ordering C
↦ [4]:
↦    // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering ds
↦ //                  : names    A B C D
↦ //        block   2 : ordering ds
↦ //                  : names    x y z u v
↦ //        block   3 : ordering C
↦ // quotient ring from ideal ...
def Px=L[1];
setring Px;
// ___ Equations of miniversal base space ___:
Js;"";
↦ Js[1,1]=BD
↦ Js[1,2]=-AD+D2
↦ Js[1,3]=CD
↦
// ___ Equations of miniversal total space ___:
Fs;"";
↦ Fs[1,1]=u2-zv-Bu+Dv
↦ Fs[1,2]=zu-yv-Au+Du
↦ Fs[1,3]=yu-xv+Cu+Dz
↦ Fs[1,4]=z2-yu-Az+By
↦ Fs[1,5]=yz-xu+Bx+Cz
↦ Fs[1,6]=y2-xz+Ax+Cy
↦
```

D.6.11.2 mod_versal

Procedure from library `deform.lib` (see Section D.6.11 [deform_lib], page 1691).

Usage: mod_versal(Mo,Io[,d,any]); Io=ideal, Mo=module, d=int, any =list

Compute: miniversal deformation of coker(Mo) over Qo=Po/Io, Po=basering;

Return: list L of 4 rings:
L[1] extending the basering Po by new variables given by "A,B,.." (deformation parameters); the new variables precede the old ones, the ordering is the product of "ls" and "ord(Po)"
L[2] = L[1]/Io extending Qo,
L[3] = the embedding ring of the versal base space,
L[4] = L[1]/(Io+Js) ring of the versal deformation of coker(Ms).
In the ring L[1] the following matrices are stored:
Js = giving the versal base space (obstructions),
Fs = giving the versal family of Mo,
Rs = giving the lifting of syzygies Lo=syz(Mo). If d is defined (!=0), it computes up to degree d.
If 'any' is defined and any[1] is no string, interactive version.
Otherwise 'any' is interpreted as a list of predefined strings: "my","param","order","out":
("my" internal prefix, "param" is a letter (e.g. "A") for the name of the first parameter or (e.g. "A(") for index parameter variables, "order" ordering string for ring extension), "out" name of output file).

Note: printlevel < 0 no additional output,
printlevel >=0,1,2,.. informs you, what is going on,
this proc uses 'execute'.

Example:

```
LIB "deform.lib";
int p = printlevel;
printlevel = 1;
ring  Ro = 0,(x,y),wp(3,4);
ideal Io = x4+y3;
matrix Mo[2][2] = x2,y,-y2,x2;
list L = mod_versal(Mo,Io);
↦ // vdim (Ext^2) = 4
↦ // vdim (Ext^1) = 4
↦ // ready: Ext1 and Ext2
↦ // Ext1 is quasi-homogeneous represented: 3,6,1,4
↦ // infinitesimal extension
↦ x2-Ax-B,   y+Cx+D,
↦ -y2+Cxy+Dy,x2+Ax+B
↦ // start deg = 2
↦ // start deg = 3
↦ // start deg = 4
↦ // start deg = 5
↦ // finished in degree
↦ 5
↦ // quasi-homogeneous weights of miniversal base
↦ 3,
↦ 6,
↦ 1,
↦ 4
↦
↦ // 'mod_versal' returned a list, say L, of four rings. In L[2] are stored\
   :
↦ //   as matrix Ms: presentation matrix of the deformed module,
↦ //   as matrix Ls: lifted syzygies,
↦ //   as matrix Js:  Equations of total space of miniversal deformation
↦ // To access these data, type
↦      def Qx=L[2]; setring Qx; print(Ms); print(Ls); print(Js);
↦
def Qx=L[2]; setring Qx;
print(Ms);
↦ x2-Ax-B+A2-C3x-3C2D+AC3,y+Cx+D,
↦ -y2+Cxy+Dy-C2x2-2CDx-D2,x2+Ax+B
print(Ls);
↦ -y-Cx-D,                x2+Ax+B,
↦ x2-Ax-B+A2-C3x-3C2D+AC3,y2-Cxy-Dy+C2x2+2CDx+D2
print(Js);
↦ -2AB+A3+3CD2-BC3-3AC2D+A2C3,
↦ -B2+A2B+D3-3BC2D+ABC3,
↦ 0,
↦ 0
printlevel = p;
if (defined(Px)) {kill Px,Qx,So;}
```

D.6.11.3 lift_kbase

Procedure from library `deform.lib` (see Section D.6.11 [deform_lib], page 1691).

Usage: lift_kbase(N,M); N,M=poly/ideal/vector/module

Return: matrix A, coefficient matrix expressing N as linear combination of k-basis of M. Let the k-basis have k elements and size(N)=c columns. Then A satisfies:
matrix(reduce(N,std(M)),k,c) = matrix(kbase(std(M)))*A

Assume: dim(M)=0 and the monomial ordering is a well ordering or the last block of the ordering is c or C

Example:

```
LIB "deform.lib";
ring R=0,(x,y),ds;
module M=[x2,xy],[y2,xy],[0,xx],[0,yy];
module N=[x3+xy,x],[x,x+y2];
print(M);
↦ x2,y2,0, 0,
↦ xy,xy,x2,y2
module kb=kbase(std(M));
print(kb);
↦ y2,xy,y,x,1,0,0,0,
↦ 0, 0, 0,0,0,y,x,1
print(N);
↦ xy+x3,x,
↦ x,    x+y2
```

```
matrix A=lift_kbase(N,M);
print(A);
↦ 0,0,
↦ 1,0,
↦ 0,0,
↦ 0,1,
↦ 0,0,
↦ 0,0,
↦ 1,1,
↦ 0,0
matrix(reduce(N,std(M)),nrows(kb),ncols(A)) - matrix(kbase(std(M)))*A;
↦ _[1,1]=0
↦ _[1,2]=0
↦ _[2,1]=0
↦ _[2,2]=0
```

D.6.11.4 lift_rel_kb

Procedure from library `deform.lib` (see Section D.6.11 [deform_lib], page 1691).

Usage: lift_rel_kb(N,M[,kbaseM,p]);

Assume: [p a monomial] or the product of all variables
N, M modules of same rank, M depending only on variables not in p and vdim(M) is finite in this ring,
[kbaseM the kbase of M in the subring given by variables not in p]
warning: these assumptions are not checked by the procedure

Return: matrix A, whose j-th columns present the coeff's of N[j] in kbaseM, i.e. kbaseM*A = reduce(N,std(M))

Example:

```
LIB "deform.lib";
ring r=0,(A,B,x,y),dp;
module M      = [x2,xy],[xy,y3],[y2],[0,x];
module kbaseM = [1],[x],[xy],[y],[0,1],[0,y],[0,y2];
poly f=xy;
module N = [AB,BBy],[A3xy+x4,AB*(1+y2)];
matrix A = lift_rel_kb(N,M,kbaseM,f);
print(A);
↦ AB,0,
↦ 0, 0,
↦ 0, A3,
↦ 0, 0,
↦ 0, AB,
↦ B2,0,
↦ 0, AB
"TEST:";
↦ TEST:
print(matrix(kbaseM)*A-matrix(reduce(N,std(M))));
↦ 0,0,
↦ 0,0
```

D.6.12 equising_lib

Library: equising.lib

Purpose: Equisingularity Stratum of a Family of Plane Curves

Author: Christoph Lossen, lossen@mathematik.uni-kl.de
Andrea Mindnich, mindnich@mathematik.uni-kl.de

Procedures:

D.6.12.1 tau_es

Procedure from library `equising.lib` (see Section D.6.12 [equising_lib], page 1696).

Usage: tau_es(f); f poly

Assume: f is a reduced bivariate polynomial, the basering has precisely two variables, is local and no qring.

Return: int, the codimension of the mu-const stratum in the semi-universal deformation base.

Note: printlevel>=1 displays additional information.
When called with any additional parameter, the computation of the Milnor number is avoided (no check for NND).

Example:

```
LIB "equising.lib";
ring r=32003,(x,y),ds;
poly f=(x4-y4)^2-x10;
tau_es(f);
↦ 42
```

See also: Section D.6.12.2 [esIdeal], page 1697; Section D.6.15.6 [invariants], page 1726; Section D.6.20.15 [tjurina], page 1763.

D.6.12.2 esIdeal

Procedure from library `equising.lib` (see Section D.6.12 [equising_lib], page 1696).

Usage: esIdeal(f[,any]]); f poly

Assume: f is a reduced bivariate polynomial, the basering has precisely two variables, is local and no qring, and the characteristic of the ground field does not divide mult(f).

Return: if called with only one parameter: list of two ideals,

_[1]: equisingularity ideal of f (in sense of Wahl),
_[2]: ideal of equisingularity with fixed position of the singularity;

if called with more than one parameter: list of three ideals,

_[1]: equisingularity ideal of f (in sense of Wahl)
_[2]: ideal of equisingularity with fixed position of the singularity;
_[3]: ideal of all g such that the deformation defined by f+eg (e^2=0) is isomorphic to an equisingular deformation of V(f) with all equimultiple sections being trivial.

Note: if some of the above condition is not satisfied then return value is list(0,0).

Example:

```
LIB "equising.lib";
ring r=0,(x,y),ds;
poly f=x7+y7+(x-y)^2*x2y2;
list K=esIdeal(f);
↦ polynomial is Newton degenerate !
↦
↦ //
↦ // versal deformation with triv. section
↦ // =====================================
↦ //
↦ //
↦ // Compute equisingularity Stratum over Spec(C[t]/t^2)
↦ // ===================================================
↦ //
↦ // finished
↦ //
option(redSB);
// Wahl's equisingularity ideal:
std(K[1]);
↦ _[1]=4x4y-10x2y3+6xy4+21x6+14y6
↦ _[2]=4x3y2-6x2y3+2xy4+7x6
↦ _[3]=x2y4-xy5
↦ _[4]=x7
↦ _[5]=xy6
↦ _[6]=y7
ring rr=0,(x,y),ds;
poly f=x4+4x3y+6x2y2+4xy3+y4+2x2y15+4xy16+2y17+xy23+y24+y30+y31;
list K=esIdeal(f);
↦ polynomial is Newton degenerate !
↦
↦ //
↦ // versal deformation with triv. section
↦ // =====================================
↦ //
↦ //
↦ // Compute equisingularity Stratum over Spec(C[t]/t^2)
↦ // ===================================================
↦ //
↦ // finished
↦ //
vdim(std(K[1]));
↦ 68
// the latter should be equal to:
tau_es(f);
↦ 68
```

See also: Section D.6.12.3 [esStratum], page 1698; Section D.6.12.1 [tau_es], page 1697.

D.6.12.3 esStratum

Procedure from library `equising.lib` (see Section D.6.12 [equising_lib], page 1696).

Usage: esStratum(F[,m,L]); F poly, m int, L list

Assume: F defines a deformation of a reduced bivariate polynomial f and the characteristic of the basering does not divide mult(f).
If nv is the number of variables of the basering, then the first nv-2 variables are the deformation parameters.
If the basering is a qring, ideal(basering) must only depend on the deformation parameters.

Compute: equations for the stratum of equisingular deformations with fixed (trivial) section.

Return: list l: either consisting of a list and an integer, where

l[1][1]=ideal defining the equisingularity stratum
l[1][2]=ideal defining the part of the equisingularity stratum where all
equimultiple sections through the non-nodes of the reduced total
transform are trivial sections
l[2]=1 if some error has occurred, l[2]=0 otherwise;

or consisting of a ring and an integer, where

l[1]=ESSring is a ring extension of basering containing the ideal ES
(describing the ES-stratum), the ideal ES_all_triv (describing the
part with trivial equimultiple sections) and the polynomial p_F=F,
l[2]=1 if some error has occurred, l[2]=0 otherwise.

Note: L is supposed to be the output of hnexpansion (with the given ordering of the variables appearing in f).
If m is given, the ES Stratum over A/maxideal(m) is computed.
This procedure uses `execute` or calls a procedure using `execute`. printlevel>=2 displays additional information.

Example:

```
LIB "equising.lib";
int p=printlevel;
printlevel=1;
ring r = 0,(a,b,c,d,e,f,g,x,y),ds;
poly F = (x2+2xy+y2+x5)+ax+by+cx2+dxy+ey2+fx3+gx4;
list M = esStratum(F);
M[1][1];
↦ _[1]=g
↦ _[2]=f
↦ _[3]=b
↦ _[4]=a
↦ _[5]=-4c+4d-4e+d2-4ce
printlevel=3;     // displays additional information
esStratum(F,2)  ; // ES-stratum over Q[a,b,c,d,e,f,g] / <a,b,c,d,e,f,g>^2
↦ //
↦ // Compute HN expansion
↦ // ---------------------
↦ // finished
↦ //
↦ // Blowup Step 1 completed
↦ // Blowup Step 2 completed
↦ // Blowup Step 3 completed
↦ // 1 branch finished
↦ //
↦ // Elimination starts:
↦ // -------------------
↦ //
↦ // Remove superfluous equations:
↦ // ----------------------------
↦ // finished
↦ //
↦ // output of 'esStratum' is a list consisting of:
↦ //    _[1][1] = ideal defining the equisingularity stratum
↦ //    _[1][2] = ideal defining the part of the equisingularity stratum
↦ //              where all equimultiple sections are trivial
↦ //    _[2] = 0
↦ [1]:
↦    [1]:
↦       _[1]=b
↦       _[2]=a
↦       _[3]=c-d+e
↦       _[4]=g
↦       _[5]=f
↦    [2]:
↦       _[1]=g
↦       _[2]=f
↦       _[3]=d-2e
↦       _[4]=c-e
↦       _[5]=b
↦       _[6]=a
↦ [2]:
↦    0
ideal I = f-fa,e+b;
qring q = std(I);
poly F = imap(r,F);
esStratum(F);
↦ //
↦ // Compute HN expansion
↦ // --------------------
↦ // finished
↦ //
↦ // Blowup Step 1 completed
↦ // Blowup Step 2 completed
↦ // Blowup Step 3 completed
↦ // 1 branch finished
↦ //
↦ // Elimination starts:
↦ // ------------------
↦ //
↦ // Remove superfluous equations:
↦ // ----------------------------
↦ // finished
↦ //
↦ // output of 'esStratum' is a list consisting of:
↦ //    _[1][1] = ideal defining the equisingularity stratum
↦ //    _[1][2] = ideal defining the part of the equisingularity stratum
↦ //              where all equimultiple sections are trivial
↦ //    _[2] = 0
↦ [1]:
```

```
↦      [1]:
↦         _[1]=e
↦         _[2]=a
↦         _[3]=-4c+4d+d2
↦         _[4]=g
↦      [2]:
↦         _[1]=g
↦         _[2]=e
↦         _[3]=d
↦         _[4]=c
↦         _[5]=a
↦   [2]:
↦      0
printlevel=p;
```

See also: Section D.6.12.2 [esIdeal], page 1697; Section D.6.12.4 [isEquising], page 1701.

D.6.12.4 isEquising

Procedure from library `equising.lib` (see Section D.6.12 [equising_lib], page 1696).

Usage: isEquising(F[,m,L]); F poly, m int, L list

Assume: F defines a deformation of a reduced bivariate polynomial f and the characteristic of the basering does not divide mult(f).
If nv is the number of variables of the basering, then the first nv-2 variables are the deformation parameters.
If the basering is a qring, ideal(basering) must only depend on the deformation parameters.

Compute: tests if the given family is equisingular along the trivial section.

Return: int: 1 if the family is equisingular, 0 otherwise.

Note: L is supposed to be the output of hnexpansion (with the given ordering of the variables appearing in f).
If m is given, the family is considered over A/maxideal(m).
This procedure uses `execute` or calls a procedure using `execute`. printlevel>=2 displays additional information.

Example:

```
LIB "equising.lib";
ring r = 0,(a,b,x,y),ds;
poly F = (x2+2xy+y2+x5)+ay3+bx5;
isEquising(F);
↦ 0
ideal I = ideal(a);
qring q = std(I);
poly F = imap(r,F);
isEquising(F);
↦ 1
ring rr=0,(A,B,C,x,y),ls;
poly f=x7+y7+(x-y)^2*x2y2;
poly F=f+A*y*diff(f,x)+B*x*diff(f,x);
isEquising(F);
↦ 0
```

```
isEquising(F,2);    // computation over  Q[a,b] / <a,b>^2
↦ 1
```

D.6.12.5 control_Matrix

Procedure from library `equising.lib` (see Section D.6.12 [equising_lib], page 1696).

Assume: L is the output of multsequence(hnexpansion(f)).

Return: list M of 4 intmat's:
M[1] contains the multiplicities at the respective infinitely near points
p[i,j] (i=step of blowup+1, j=branch) – if branches j=k,...,k+m pass
through the same p[i,j] then the multiplicity is stored in M[1][k,j],
while M[1][k+1]=...=M[1][k+m]=0.
M[2] contains the number of branches meeting at p[i,j] (again, the information
is stored according to the above rule)
M[3] contains the information about the splitting of M[1][i,j] with respect to
different tangents of branches at p[i,j] (information is stored only for
minimal j>=k corresponding to a new tangent direction).
The entries are the sum of multiplicities of all branches with the
respective tangent.
M[4] contains the maximal sum of higher multiplicities for a branch passing
through p[i,j] (= degree Bound for blowing up)

Note: the branches are ordered in such a way that only consecutive branches can meet at an infinitely near point.
the final rows of the matrices M[1],...,M[3] is (1,1,1,...,1), and correspond to infinitely near points such that the strict transforms of the branches are smooth and intersect the exceptional divisor transversally.

See also: Section D.6.15.8 [multsequence], page 1727.

D.6.13 gmssing_lib

Library: gmssing.lib

Purpose: Gauss-Manin System of Isolated Singularities

Author: Mathias Schulze, mschulze at mathematik.uni-kl.de

Overview: A library for computing invariants related to the Gauss-Manin system of an isolated hypersurface singularity.

References:

[Sch01] M. Schulze: Algorithms for the Gauss-Manin connection. J. Symb. Comp. 32,5 (2001), 549-564.
[Sch02] M. Schulze: The differential structure of the Brieskorn lattice. In: A.M. Cohen et al.: Mathematical Software - ICMS 2002. World Scientific (2002).
[Sch03] M. Schulze: Monodromy of Hypersurface Singularities. Acta Appl. Math. 75 (2003), 3-13.
[Sch04] M. Schulze: A normal form algorithm for the Brieskorn lattice. J. Symb. Comp. 38,4 (2004), 1207-1225.

Procedures: See also: Section 7.5.4 [dmod_lib], page 400; Section D.6.14 [gmspoly_lib], page 1717; Section D.6.17 [mondromy_lib], page 1737; Section D.6.22 [spectrum_lib], page 1772.

D.6.13.1 gmsring

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: gmsring(t,s); poly t, string s

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

ring G; Gauss-Manin system of t with variable s
  poly gmspoly=t;
  ideal gmsjacob; Jacobian ideal of t
  ideal gmsstd; standard basis of Jacobian ideal
  matrix gmsmatrix; matrix(gmsjacob)*gmsmatrix==matrix(gmsstd)
  ideal gmsbasis; monomial vector space basis of Jacobian algebra
  int Gmssing::gmsmaxdeg; maximal weight of variables

Note: gmsbasis is a C[[s]]-basis of H" and [t,s]=s^2

Example:

```
LIB "gmssing.lib";
ring @R=0,(x,y),ds;
poly t=x5+x2y2+y5;
def G=gmsring(t,"s");
setring(G);
gmspoly;
↦ x2y2+x5+y5
print(gmsjacob);
↦ 2xy2+5x4,
↦ 2x2y+5y4
print(gmsstd);
↦ 2x2y+5y4,
↦ 5x5-5y5,
↦ 2xy2+5x4,
↦ 10y6+25x3y4
print(gmsmatrix);
↦ 0,x, 1,-2xy,
↦ 1,-y,0,2y2+5x3
print(gmsbasis);
↦ y5,
↦ y4,
↦ y3,
↦ y2,
↦ xy,
↦ y,
↦ x4,
↦ x3,
↦ x2,
↦ x,
↦ 1
Gmssing::gmsmaxdeg;
↦ 1
```

D.6.13.2 gmsnf

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: gmsnf(p,K); poly p, int K

Assume: basering returned by gmsring

Return: list nf;
ideal nf[1]; projection of p to <gmsbasis>C[[s]] mod s^(K+1) ideal nf[2]; p==nf[1]+nf[2]

Note: computation can be continued by setting p=nf[2]

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
def G=gmsring(t,"s");
setring(G);
list l0=gmsnf(gmspoly,0);
print(l0[1]);
↦ -1/2y5
list l1=gmsnf(gmspoly,1);
print(l1[1]);
↦ -1/2y5+1/2s
list l=gmsnf(l0[2],1);
print(l[1]);
↦ 1/2s
```

D.6.13.3 gmscoeffs

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: gmscoeffs(p,K); poly p, int K

Assume: basering constructed by gmsring

Return:

list l;
  matrix l[1]; C[[s]]-basis representation of p mod s^(K+1)
  ideal l[2]; p==matrix(gmsbasis)*l[1]+l[2]

Note: computation can be continued by setting p=l[2]

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
def G=gmsring(t,"s");
setring(G);
list l0=gmscoeffs(gmspoly,0);
print(l0[1]);
↦ -1/2,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0
list l1=gmscoeffs(gmspoly,1);
print(l1[1]);
↦ -1/2,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 1/2s
list l=gmscoeffs(l0[2],1);
print(l[1]);
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 0,
↦ 1/2s
```

D.6.13.4 bernstein

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing lib], page 1702).

Usage: bernstein(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list bs; Bernstein-Sato polynomial b(s) of t
  ideal bs[1];
    number bs[1][i]; i-th root of b(s)
  intvec bs[2];
    int bs[2][i]; multiplicity of i-th root of b(s)

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
bernstein(t);
↦ [1]:
```

```
↦     _[1]=-13/10
↦     _[2]=-11/10
↦     _[3]=-1
↦     _[4]=-9/10
↦     _[5]=-7/10
↦     _[6]=-1/2
↦  [2]:
↦     1,1,2,1,1,2
```

D.6.13.5 monodromy

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: monodromy(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list l; Jordan data jordan(M) of monodromy matrix exp(-2*pi*i*M)
  ideal l[1];
    number l[1][i]; eigenvalue of i-th Jordan block of M
  intvec l[2];
    int l[2][i]; size of i-th Jordan block of M
  intvec l[3];
    int l[3][i]; multiplicity of i-th Jordan block of M

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
monodromy(t);
↦  [1]:
↦     _[1]=1/2
↦     _[2]=7/10
↦     _[3]=9/10
↦     _[4]=1
↦     _[5]=11/10
↦     _[6]=13/10
↦  [2]:
↦     2,1,1,1,1,1
↦  [3]:
↦     1,2,2,1,2,2
```

See also: Section D.3.2 [linalg_lib], page 987; Section D.6.17 [mondromy_lib], page 1737.

D.6.13.6 spectrum

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spectrum(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list sp; singularity spectrum of t
  ideal sp[1];
    number sp[1][i]; i-th spectral number
  intvec sp[2];
    int sp[2][i]; multiplicity of i-th spectral number

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
spprint(spectrum(t));
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
```

See also: Section D.6.22 [spectrum_lib], page 1772.

D.6.13.7 sppairs

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: sppairs(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list spp; spectral pairs of t
  ideal spp[1];
    number spp[1][i]; V-filtration index of i-th spectral pair
  intvec spp[2];
    int spp[2][i]; weight filtration index of i-th spectral pair
  intvec spp[3];
    int spp[3][i]; multiplicity of i-th spectral pair

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
sppprint(sppairs(t));
↦ ((-1/2,2),1),((-3/10,1),2),((-1/10,1),2),((0,1),1),((1/10,1),2),((3/10,1)\
   ,2),((1/2,0),1)
```

See also: Section D.6.22 [spectrum_lib], page 1772.

D.6.13.8 vfilt

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: vfilt(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list v; V-filtration on H"/s*H"
  ideal v[1];
    number v[1][i]; V-filtration index of i-th spectral number
  intvec v[2];

int v[2][i]; multiplicity of i-th spectral number
list v[3];
module v[3][i]; vector space of i-th graded part in terms of v[4]
ideal v[4]; monomial vector space basis of H”/s*H”
ideal v[5]; standard basis of Jacobian ideal

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
vfilt(t);
↦ [1]:
↦    _[1]=-1/2
↦    _[2]=-3/10
↦    _[3]=-1/10
↦    _[4]=0
↦    _[5]=1/10
↦    _[6]=3/10
↦    _[7]=1/2
↦ [2]:
↦    1,2,2,1,2,2,1
↦ [3]:
↦    [1]:
↦       _[1]=gen(11)
↦    [2]:
↦       _[1]=gen(10)
↦       _[2]=gen(6)
↦    [3]:
↦       _[1]=gen(9)
↦       _[2]=gen(4)
↦    [4]:
↦       _[1]=gen(5)
↦    [5]:
↦       _[1]=gen(3)
↦       _[2]=gen(8)
↦    [6]:
↦       _[1]=gen(2)
↦       _[2]=gen(7)
↦    [7]:
↦       _[1]=gen(1)
↦ [4]:
↦    _[1]=y5
↦    _[2]=y4
↦    _[3]=y3
↦    _[4]=y2
↦    _[5]=xy
↦    _[6]=y
↦    _[7]=x4
↦    _[8]=x3
↦    _[9]=x2
↦    _[10]=x
↦    _[11]=1
↦ [5]:
```

```
↦     _[1]=2x2y+5y4
↦     _[2]=5x5-5y5
↦     _[3]=2xy2+5x4
↦     _[4]=10y6+25x3y4
```

See also: Section D.6.22 [spectrum_lib], page 1772.

D.6.13.9 vwfilt

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: vwfilt(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

```
list vw;  weighted V-filtration on H”/s*H”
  ideal vw[1];
    number vw[1][i];  V-filtration index of i-th spectral pair
  intvec vw[2];
    int vw[2][i];  weight filtration index of i-th spectral pair
  intvec vw[3];
    int vw[3][i];  multiplicity of i-th spectral pair
  list vw[4];
    module vw[4][i];  vector space of i-th graded part in terms of vw[5]
  ideal vw[5];  monomial vector space basis of H”/s*H”
  ideal vw[6];  standard basis of Jacobian ideal
```

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
vwfilt(t);
↦ [1]:
↦     _[1]=-1/2
↦     _[2]=-3/10
↦     _[3]=-1/10
↦     _[4]=0
↦     _[5]=1/10
↦     _[6]=3/10
↦     _[7]=1/2
↦ [2]:
↦     2,1,1,1,1,1,0
↦ [3]:
↦     1,2,2,1,2,2,1
↦ [4]:
↦     [1]:
↦        _[1]=gen(11)
↦     [2]:
↦        _[1]=gen(10)
↦        _[2]=gen(6)
↦     [3]:
↦        _[1]=gen(9)
↦        _[2]=gen(4)
↦      [4]:
↦         _[1]=gen(5)
↦      [5]:
↦         _[1]=gen(3)
↦         _[2]=gen(8)
↦      [6]:
↦         _[1]=gen(2)
↦         _[2]=gen(7)
↦      [7]:
↦         _[1]=gen(1)
↦   [5]:
↦      _[1]=y5
↦      _[2]=y4
↦      _[3]=y3
↦      _[4]=y2
↦      _[5]=xy
↦      _[6]=y
↦      _[7]=x4
↦      _[8]=x3
↦      _[9]=x2
↦      _[10]=x
↦      _[11]=1
↦   [6]:
↦      _[1]=2x2y+5y4
↦      _[2]=5x5-5y5
↦      _[3]=2xy2+5x4
↦      _[4]=10y6+25x3y4
```

See also: Section D.6.22 [spectrum_lib], page 1772.

D.6.13.10 tmatrix

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: tmatrix(t); poly t

Assume: characteristic 0; local degree ordering;
isolated critical point 0 of t

Return:

list l=A0,A1,T,M;
  matrix A0,A1; t=A0+s*A1+s^2*(d/ds) on H" w.r.t. C[[s]]-basis M*T
  module T; C-basis of C^mu
  ideal M; monomial C-basis of H"/sH"

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
list l=tmatrix(t);
print(l[1]);
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 1,0,0,0,0,0,0,0,0,0,0
print(l[2]);
↦ 1/2,0,   0,   0,   0,   0,0,    0,    0,    0,    0,
↦ 0,  7/10,0,   0,   0,   0,0,    0,    0,    0,    0,
↦ 0,  0,   7/10,0,   0,   0,0,    0,    0,    0,    0,
↦ 0,  0,   0,   9/10,0,   0,0,    0,    0,    0,    0,
↦ 0,  0,   0,   0,   9/10,0,0,    0,    0,    0,    0,
↦ 0,  0,   0,   0,   0,   1,0,    0,    0,    0,    0,
↦ 0,  0,   0,   0,   0,   0,11/10,0,    0,    0,    0,
↦ 0,  0,   0,   0,   0,   0,0,    11/10,0,    0,    0,
↦ 0,  0,   0,   0,   0,   0,0,    0,    13/10,0,    0,
↦ 0,  0,   0,   0,   0,   0,0,    0,    0,    13/10,0,
↦ 0,  0,   0,   0,   0,   0,0,    0,    0,    0,    3/2
print(l[3]);
↦ 85/4,0,  0,  0,0,85/8,0,0,0,0,1/2,
↦ 0,   125,0,  0,0,0,   0,0,1,0,0,
↦ 0,   0,  0,  5,0,0,   1,0,0,0,0,
↦ 0,   0,  0,  0,4,0,   0,0,0,0,0,
↦ 2,   0,  0,  0,0,1,   0,0,0,0,0,
↦ 0,   0,  16, 0,0,0,   0,0,0,0,0,
↦ 0,   0,  125,0,0,0,   0,0,0,1,0,
↦ 0,   0,  0,  0,5,0,   0,1,0,0,0,
↦ 0,   0,  0,  4,0,0,   0,0,0,0,0,
↦ 0,   16, 0,  0,0,0,   0,0,0,0,0,
↦ -1,  0,  0,  0,0,0,   0,0,0,0,0
print(l[4]);
↦ y5,
↦ y4,
↦ y3,
↦ y2,
↦ xy,
↦ y,
↦ x4,
↦ x3,
↦ x2,
↦ x,
↦ 1
```

D.6.13.11 endvfilt

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: endvfilt(v); list v

Assume: v returned by vfilt

Return:

list ev; V-filtration on Jacobian algebra

ideal ev[1];
  number ev[1][i]; i-th V-filtration index
intvec ev[2];
  int ev[2][i]; i-th multiplicity
list ev[3];
  module ev[3][i]; vector space of i-th graded part in terms of ev[4]
ideal ev[4]; monomial vector space basis of Jacobian algebra
ideal ev[5]; standard basis of Jacobian ideal

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
poly t=x5+x2y2+y5;
endvfilt(vfilt(t));
↦ [1]:
↦    _[1]=0
↦    _[2]=1/5
↦    _[3]=2/5
↦    _[4]=1/2
↦    _[5]=3/5
↦    _[6]=4/5
↦    _[7]=1
↦ [2]:
↦    1,2,2,1,2,2,1
↦ [3]:
↦    [1]:
↦       _[1]=gen(11)
↦    [2]:
↦       _[1]=gen(10)
↦       _[2]=gen(6)
↦    [3]:
↦       _[1]=gen(9)
↦       _[2]=gen(4)
↦    [4]:
↦       _[1]=gen(5)
↦    [5]:
↦       _[1]=gen(8)
↦       _[2]=gen(3)
↦    [6]:
↦       _[1]=gen(7)
↦       _[2]=gen(2)
↦    [7]:
↦       _[1]=gen(1)
↦ [4]:
↦    _[1]=y5
↦    _[2]=y4
↦    _[3]=y3
↦    _[4]=y2
↦    _[5]=xy
↦    _[6]=y
↦    _[7]=x4
↦    _[8]=x3
↦    _[9]=x2
↦     _[10]=x
↦     _[11]=1
↦  [5]:
↦     _[1]=2x2y+5y4
↦     _[2]=5x5-5y5
↦     _[3]=2xy2+5x4
↦     _[4]=10y6+25x3y4
```

D.6.13.12 sppnf

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: sppnf(list(a,w[,m])); ideal a, intvec w, intvec m

Assume: ncols(a)==size(w)==size(m)

Return: order (a[i][,w[i]]) with multiplicity m[i] lexicographically

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-3/10,-1/10,-1/10,0,1/10,1/10,3/10,3/10,1/2),
intvec(2,1,1,1,1,1,1,1,1,1,0));
sppprint(sppnf(sp));
↦ ((-1/2,2),1),((-3/10,1),2),((-1/10,1),2),((0,1),1),((1/10,1),2),((3/10,1)\
   ,2),((1/2,0),1)
```

D.6.13.13 sppprint

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: sppprint(spp); list spp

Return: string s; spectral pairs spp

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list spp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(2,1,1,1,1,1,0),
intvec(1,2,2,1,2,2,1));
sppprint(spp);
↦ ((-1/2,2),1),((-3/10,1),2),((-1/10,1),2),((0,1),1),((1/10,1),2),((3/10,1)\
   ,2),((1/2,0),1)
```

D.6.13.14 spadd

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spadd(sp1,sp2); list sp1, list sp2

Return: list sp; sum of spectra sp1 and sp2

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp1=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp1);
```

```
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
list sp2=list(ideal(-1/6,1/6),intvec(1,1));
spprint(sp2);
↦ (-1/6,1),(1/6,1)
spprint(spadd(sp1,sp2));
↦ (-1/2,1),(-3/10,2),(-1/6,1),(-1/10,2),(0,1),(1/10,2),(1/6,1),(3/10,2),(1/\
   2,1)
```

D.6.13.15 spsub

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spsub(sp1,sp2); list sp1, list sp2

Return: list sp; difference of spectra sp1 and sp2

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp1=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp1);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
list sp2=list(ideal(-1/6,1/6),intvec(1,1));
spprint(sp2);
↦ (-1/6,1),(1/6,1)
spprint(spsub(sp1,sp2));
↦ (-1/2,1),(-3/10,2),(-1/6,-1),(-1/10,2),(0,1),(1/10,2),(1/6,-1),(3/10,2),(\
   1/2,1)
```

D.6.13.16 spmul

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spmul(sp0,k); list sp0, int[vec] k

Return: list sp; linear combination of spectra sp0 with coefficients k

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
spprint(spmul(sp,2));
↦ (-1/2,2),(-3/10,4),(-1/10,4),(0,2),(1/10,4),(3/10,4),(1/2,2)
list sp1=list(ideal(-1/6,1/6),intvec(1,1));
spprint(sp1);
↦ (-1/6,1),(1/6,1)
list sp2=list(ideal(-1/3,0,1/3),intvec(1,2,1));
spprint(sp2);
↦ (-1/3,1),(0,2),(1/3,1)
spprint(spmul(list(sp1,sp2),intvec(1,2)));
↦ (-1/3,2),(-1/6,1),(0,4),(1/6,1),(1/3,2)
```

D.6.13.17 spissemicont

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spissemicont(sp[,1]); list sp, int opt

Return:

int k=
  1; if sum of sp is positive on all intervals [a,a+1) [and (a,a+1)]
  0; if sum of sp is negative on some interval [a,a+1) [or (a,a+1)]

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp1=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp1);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
list sp2=list(ideal(-1/6,1/6),intvec(1,1));
spprint(sp2);
↦ (-1/6,1),(1/6,1)
spissemicont(spsub(sp1,spmul(sp2,3)));
↦ 1
spissemicont(spsub(sp1,spmul(sp2,4)));
↦ 0
```

D.6.13.18 spsemicont

Procedure from library `gmssing.lib` (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spsemicont(sp0,sp,k[,1]); list sp0, list sp

Return:

list l;
  intvec l[i]; if the spectra sp0 occur with multiplicities k
    in a deformation of a [quasihomogeneous] singularity
    with spectrum sp then k<=l[i]

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp0=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp0);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
list sp1=list(ideal(-1/6,1/6),intvec(1,1));
spprint(sp1);
↦ (-1/6,1),(1/6,1)
list sp2=list(ideal(-1/3,0,1/3),intvec(1,2,1));
spprint(sp2);
↦ (-1/3,1),(0,2),(1/3,1)
list sp=sp1,sp2;
list l=spsemicont(sp0,sp);
l;
↦ [1]:
↦    3
↦ [2]:
```

```
↦    2,1
spissemicont(spsub(sp0,spmul(sp,l[1])));
↦ 1
spissemicont(spsub(sp0,spmul(sp,l[1]-1)));
↦ 1
spissemicont(spsub(sp0,spmul(sp,l[1]+1)));
↦ 0
```

D.6.13.19 spmilnor

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spmilnor(sp); list sp

Return: int mu; Milnor number of spectrum sp

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
spmilnor(sp);
↦ 11
```

D.6.13.20 spgeomgenus

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spgeomgenus(sp); list sp

Return: int g; geometrical genus of spectrum sp

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
spgeomgenus(sp);
↦ 6
```

D.6.13.21 spgamma

Procedure from library gmssing.lib (see Section D.6.13 [gmssing_lib], page 1702).

Usage: spgamma(sp); list sp

Return: number gamma; gamma invariant of spectrum sp

Example:

```
LIB "gmssing.lib";
ring R=0,(x,y),ds;
list sp=list(ideal(-1/2,-3/10,-1/10,0,1/10,3/10,1/2),intvec(1,2,2,1,2,2,1));
spprint(sp);
↦ (-1/2,1),(-3/10,2),(-1/10,2),(0,1),(1/10,2),(3/10,2),(1/2,1)
spgamma(sp);
↦ 1/240
```

D.6.14 gmspoly_lib

Library: gmspoly.lib

Purpose: Gauss-Manin System of Tame Polynomials

Author: Mathias Schulze, mschulze at mathematik.uni-kl.de

Overview: A library for computing the Gauss-Manin system of a cohomologically tame polynomial f. Schulze's algorithm [Sch05], based on Sabbah's theory [Sab98], is used to compute a good basis of (the Brieskorn lattice of) the Gauss-Manin system and the differential operation of f in terms of this basis. In addition, there is a test for tameness in the sense of Broughton. Tame polynomials can be considered as an affine algebraic analogue of local analytic isolated hypersurface singularities. They have only finitely many citical points, and those at infinity do not give rise to atypical values in a sense depending on the precise notion of tameness considered. Well-known notions of tameness like tameness, M-tameness, Malgrange-tameness, and cohomological tameness, and their relations, are reviewed in [Sab98,8]. For ordinary tameness, see Broughton [Bro88,3].
Sabbah [Sab98] showed that the Gauss-Manin system, the D-module direct image of the structure sheaf, of a cohomologically tame polynomial carries a similar structure as in the isolated singularity case, coming from a Mixed Hodge structure on the cohomology of the Milnor (typical) fibre (see gmssing.lib). The data computed by this library encodes the differential structure of the Gauss-Manin system, and the Mixed Hodge structure of the Milnor fibre over the complex numbers. As a consequence, it yields the Hodge numbers, spectral pairs, and monodromy at infinity.

References:

[Bro88] S. Broughton: Milnor numbers and the topology of polynomial hypersurfaces. Inv. Math. 92 (1988) 217-241.
[Sab98] C. Sabbah: Hypergeometric periods for a tame polynomial. arXiv.org math.AG/9805077.
[Sch05] M. Schulze: Good bases for tame polynomials. J. Symb. Comp. 39,1 (2005), 103-126.

Procedures: See also: Section D.6.13 [gmssing_lib], page 1702.

D.6.14.1 isTame

Procedure from library `gmspoly.lib` (see Section D.6.14 [gmspoly_lib], page 1717).

Usage: isTame(f); poly f

Assume: basering has no variables named w(1),w(2),...

Return:

int k=
  1; if f is tame in the sense of Broughton [Bro88,3]
  0; if f is not tame

Remarks: procedure implements Proposition 3.1 in [Bro88]

Example:

```
LIB "gmspoly.lib";
ring R=0,(x,y),dp;
isTame(x2y+x);
↦ 0
isTame(x3+y3+xy);
↦ 1
```

D.6.14.2 goodBasis

Procedure from library `gmspoly.lib` (see Section D.6.14 [gmspoly_lib], page 1717).

Usage: goodBasis(f); poly f

Assume: f is cohomologically tame in the sense of Sabbah [Sab98,8]

Return:

ring R; basering with new variable s
ideal b; [matrix(b)] is a good basis of the Brieskorn lattice
matrix A; A(s)=A0+s*A1 and t[matrix(b)]=[matrix(b)](A(s)+s^2*(d/ds))

Remarks: procedure implements Algorithm 6 in [Sch05]

Example:

```
LIB "gmspoly.lib";
ring R=0,(x,y,z),dp;
poly f=x+y+z+x2y2z2;
def Rs=goodBasis(f);
setring(Rs);
b;
↦ b[1]=1
↦ b[2]=s2x-3sx2+x3
↦ b[3]=5/2x
↦ b[4]=10s2x2-25/2sx3+5/2x4
↦ b[5]=-25/4sx+25/4x2
print(jet(A,0));
↦ 0,0,0,-25/8,0,
↦ 0,0,0,0,    125/8,
↦ 1,0,0,0,    0,
↦ 0,1,0,0,    0,
↦ 0,0,1,0,    0
print(jet(A/var(1),0));
↦ 1/2,0,0,  0,0,
↦ 0,  1,0,  0,0,
↦ 0,  0,3/2,0,0,
↦ 0,  0,0,  2,0,
↦ 0,  0,0,  0,5/2
```

See also: Section D.6.13 [gmssing_lib], page 1702.

D.6.15 hnoether_lib

Library: hnoether.lib

Purpose: Hamburger-Noether (Puiseux) Expansion

Authors: Martin Lamm, lamm@mathematik.uni-kl.de
Christoph Lossen, lossen@mathematik.uni-kl.de

Overview: A library for computing the Hamburger-Noether expansion (analogue of Puiseux expansion over fields of arbitrary characteristic) of a reduced plane curve singularity following [Campillo, A.: Algebroid curves in positive characteristic, Springer LNM 813 (1980)].
The library contains also procedures for computing the (topological) numerical invariants of plane curve singularities.

Procedures:

D.6.15.1 hnexpansion

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: hnexpansion(f[,"ess"]); f poly

Assume: f is a bivariate polynomial (in the first 2 ring variables)

Return: list `L`, containing Hamburger-Noether data of `f`: If the computation of the HNE required no field extension, `L` is a list of lists `L[i]` (corresponding to the output of `develop`, applied to a branch of `f`, but the last entry being omitted):

`L[i][1]`; matrix:
: Each row contains the coefficients of the corresponding line of the Hamburger-Noether expansion (HNE) for the i-th branch. The end of the line is marked in the matrix by the first ring variable (usually x).

`L[i][2]`; intvec:
: indicating the length of lines of the HNE

`L[i][3]`; int:
: 0 if the 1st ring variable was transversal (with respect to the i-th branch),
1 if the variables were changed at the beginning of the computation,
-1 if an error has occurred.

`L[i][4]`; poly:
: the transformed equation of the i-th branch to make it possible to extend the Hamburger-Noether data a posteriori without having to do all the previous calculation once again (0 if not needed).

If the computation of the HNE required a field extension, the first entry `L[1]` of the list is a ring, in which a list `hne` of lists (the HN data, as above) and a polynomial `f` (image of `f` over the new field) are stored.
If called with an additional input parameter, `hnexpansion` computes only one representative for each class of conjugate branches (over the ground field active when calling the procedure). In this case, the returned list `L` always has only two entries: `L[1]` is either a list of lists (the HN data) or a ring (as above), and `L[2]` is an integer vector (the number of branches in the respective conjugacy classes).

Note: If f is known to be irreducible as a power series, `develop(f)` could be chosen instead to avoid a change of basering during the computations.
Increasing `printlevel` leads to more and more comments.
Having defined a variable `HNDebugOn` leads to a maximum number of comments.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
// First, an example which requires no field extension:
list Hne=hnexpansion(x4-y6);
↦ // No change of ring necessary, return value is HN expansion.
size(Hne);           // number of branches
↦ 2
displayHNE(Hne);     // HN expansion of branches
↦ // Hamburger-Noether development of branch nr.1:
↦   x = z(1)*y
↦   y = z(1)^2
↦
```

```
↦ // Hamburger-Noether development of branch nr.2:
↦   x = z(1)*y
↦   y = -z(1)^2
↦
param(Hne[1]);       // parametrization of 1st branch
↦ _[1]=x3
↦ _[2]=x2
param(Hne[2]);       // parametrization of 2nd branch
↦ _[1]=-x3
↦ _[2]=-x2
// An example which requires a field extension:
list L=hnexpansion((x4-y6)*(y2+x4));
↦
↦ // 'hnexpansion' created a list of one ring.
↦ // To see the ring and the data stored in the ring, type (if you assigned
↦ // the name L to the list):
↦      show(L);
↦ // To display the computed HN expansion, type
↦      def HNring = L[1]; setring HNring;  displayHNE(hne);
def R=L[1]; setring R; displayHNE(hne);
↦ // Hamburger-Noether development of branch nr.1:
↦   y = (a)*x^2
↦
↦ // Hamburger-Noether development of branch nr.2:
↦   y = (-a)*x^2
↦
↦ // Hamburger-Noether development of branch nr.3:
↦   x = z(1)*y
↦   y = z(1)^2
↦
↦ // Hamburger-Noether development of branch nr.4:
↦   x = z(1)*y
↦   y = -z(1)^2
↦
basering;
↦ // coefficients: QQ[a]/(a2+1)
↦ // number of vars : 2
↦ //        block   1 : ordering ls
↦ //                  : names    x y
↦ //        block   2 : ordering C
setring r; kill R;
// Computing only one representative per conjugacy class:
L=hnexpansion((x4-y6)*(y2+x4),"ess");
↦
↦ // 'hnexpansion' created a list of one ring.
↦ // To see the ring and the data stored in the ring, type (if you assigned
↦ // the name L to the list):
↦      show(L);
↦ // To display the computed HN expansion, type
↦      def HNring = L[1]; setring HNring;  displayHNE(hne);
↦ // As second entry of the returned list L, you obtain an integer vector,
↦ // indicating the number of conjugates for each of the computed branches.
def R=L[1]; setring R; displayHNE(hne);
```

```
↦ // Hamburger-Noether development of branch nr.1:
↦   y = (a)*x^2
↦
↦ // Hamburger-Noether development of branch nr.2:
↦   x = z(1)*y
↦   y = z(1)^2
↦
↦ // Hamburger-Noether development of branch nr.3:
↦   x = z(1)*y
↦   y = -z(1)^2
↦
L[2];     // number of branches in respective conjugacy classes
↦ 2,1,1
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.5 [displayHNE], page 1725; Section D.6.15.3 [extdevelop], page 1722; Section D.6.15.4 [param], page 1723.

D.6.15.2 develop

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: develop(f [,n]); f poly, n int

Assume: f is a bivariate polynomial (in the first 2 ring variables) and irreducible as power series (for reducible f use `hnexpansion`).

Return: list `L` with:

`L[1]`; matrix:
Each row contains the coefficients of the corresponding line of the Hamburger-Noether expansion (HNE). The end of the line is marked in the matrix by the first ring variable (usually x).

`L[2]`; intvec:
indicating the length of lines of the HNE

`L[3]`; int: 0 if the 1st ring variable was transversal (with respect to f),
1 if the variables were changed at the beginning of the computation,
-1 if an error has occurred.

`L[4]`; poly:
the transformed polynomial of f to make it possible to extend the Hamburger-Noether development a posteriori without having to do all the previous calculation once again (0 if not needed)

`L[5]`; int: 1 if the curve has exactly one branch (i.e., is irreducible),
0 else (i.e., the curve has more than one HNE, or f is not valid).

Display: The (non zero) elements of the HNE (if not called by another proc).

Note: The optional parameter `n` affects only the computation of the LAST line of the HNE. If it is given, the HN-matrix `L[1]` will have at least `n` columns.
Otherwise, the number of columns will be chosen minimal such that the matrix contains all necessary information (i.e., all lines of the HNE but the last (which is in general infinite) have place).
If `n` is negative, the algorithm is stopped as soon as the computed information is sufficient for `invariants(L)`, but the HN-matrix `L[1]` may still contain undetermined elements, which are marked with the 2nd variable (of the basering).

For time critical computations it is recommended to use `ring ...,(x,y),ls` as basering - it increases the algorithm's speed.
If `printlevel>=0` comments are displayed (default is `printlevel=0`).

Example:

```
LIB "hnoether.lib";
ring exring = 7,(x,y),ds;
list Hne=develop(4x98+2x49y7+x11y14+2y14);
print(Hne[1]);
↦ 0,0, 0,0,0,0,3,x,
↦ 0,x, 0,0,0,0,0,0,
↦ 0,0, 0,x,0,0,0,0,
↦ 0,x, 0,0,0,0,0,0,
↦ 0,-1,0,0,0,0,0,0
// therefore the HNE is:
// z(-1)= 3*z(0)^7 + z(0)^7*z(1),
// z(0) = z(1)*z(2),       (there is 1 zero in the 2nd row before x)
// z(1) = z(2)^3*z(3),     (there are 3 zeroes in the 3rd row)
// z(2) = z(3)*z(4),
// z(3) = -z(4)^2 + 0*z(4)^3 +...+ 0*z(4)^8 + ?*z(4)^9 + ...
// (the missing x in the last line indicates that it is not complete.)
Hne[2];
↦ 7,1,3,1,-1
param(Hne);
↦ // ** Warning: result is exact up to order 20 in x and 104 in y !
↦ _[1]=-x14
↦ _[2]=-3x98-x109
// parametrization:   x(t)= -t^14+O(t^21),  y(t)= -3t^98+O(t^105)
// (the term -t^109 in y may have a wrong coefficient)
displayHNE(Hne);
↦   y = 3*x^7+z(1)*x^7
↦   x = z(1)*z(2)
↦   z(1) = z(2)^3*z(3)
↦   z(2) = z(3)*z(4)
↦   z(3) = -z(4)^2 + ..... (terms of degree >=9)
```

See also: Section D.6.15.5 [displayHNE], page 1725; Section D.6.15.3 [extdevelop], page 1722; Section D.6.15.1 [hnexpansion], page 1719.

D.6.15.3 extdevelop

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: extdevelop(L,N); list L, int N

Assume: L is the output of `develop(f)`, or of `extdevelop(l,n)`, or one entry in the list `hne` in the ring created by `hnexpansion(f[,"ess"])`.

Return: an extension of the Hamburger-Noether development of f as a list in the same format as L has (up to the last entry in the output of `develop(f)`).
Type `help develop;`, resp. `help hnexpansion;` for more details.

Note: The new HN-matrix will have at least N columns (if the HNE is not finite). In particular, if f is irreducible then (in most cases) `extdevelop(develop(f),N)` will produce the same result as `develop(f,N)`.
If the matrix M of L has n columns then, compared with `parametrization(L)`,

`paramametrize(extdevelop(L,N))` will increase the exactness by at least (N-n) more significant monomials.

Example:

```
LIB "hnoether.lib";
ring exring=0,(x,y),dp;
list Hne=hnexpansion(x14-3y2x11-y3x10-y2x9+3y4x8+y5x7+3y4x6+x5*(-y6+y5)
-3y6x3-y7x2+y8);
↦ // No change of ring necessary, return value is HN expansion.
displayHNE(Hne);    // HNE of 1st,3rd branch is finite
↦ // Hamburger-Noether development of branch nr.1:
↦   y = z(1)*x
↦   x = z(1)^2
↦
↦ // Hamburger-Noether development of branch nr.2:
↦   y = z(1)*x
↦   x = z(1)^2+z(1)^2*z(2)
↦   z(1) = z(2)^2-z(2)^3 + ..... (terms of degree >=4)
↦
↦ // Hamburger-Noether development of branch nr.3:
↦   y = z(1)*x^2
↦   x = z(1)^2
↦
print(extdevelop(Hne[1],5)[1]);
↦  No extension is possible
↦ 0,x,0,
↦ 0,1,x
list ehne=extdevelop(Hne[2],5);
displayHNE(ehne);
↦   y = z(1)*x
↦   x = z(1)^2+z(1)^2*z(2)
↦   z(1) = z(2)^2-z(2)^3+z(2)^4-z(2)^5 + ..... (terms of degree >=6)
param(Hne[2]);
↦ // ** Warning: result is exact up to order 5 in x and 7 in y !
↦ _[1]=x7-x6-x5+x4
↦ _[2]=-x10+2x9-2x7+x6
param(ehne);
↦ // ** Warning: result is exact up to order 7 in x and 9 in y !
↦ _[1]=x11-x10+x9-x8-x7+x6-x5+x4
↦ _[2]=-x16+2x15-3x14+4x13-2x12+2x10-4x9+3x8-2x7+x6
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.4 [param], page 1723.

D.6.15.4 param

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: param(L [,s]); L list, s any type (optional)

Assume: L is the output of `develop(f)`, or of `extdevelop(develop(f),n)`, or (one entry in) the list of HN data created by `hnexpansion(f[,"ess"])`.

Return: If L are the HN data of an irreducible plane curve singularity f: a parametrization for f in the following format:

- if only the list L is given, the result is an ideal of two polynomials p[1],p[2]: if the HNE was finite then f(p[1],p[2])=0}; if not, the true parametrization will be given by two power series, and p[1],p[2] are truncations of these series.
- if the optional parameter s is given, the result is a list l: l[1]=param(L) (ideal) and l[2]=intvec with two entries indicating the highest degree up to which the coefficients of the monomials in l[1] are exact (entry -1 means that the corresponding parametrization is exact).
If L collects the HN data of a reducible plane curve singularity f, the return value is a list of parametrizations in the respective format.

Note: If the basering has only 2 variables, the first variable is chosen as indefinite. Otherwise, the 3rd variable is chosen.

Example:

```
LIB "hnoether.lib";
ring exring=0,(x,y,t),ds;
poly f=x3+2xy2+y2;
list Hne=develop(f);
list hne_extended=extdevelop(Hne,10);
//   compare the HNE matrices ...
print(Hne[1]);
↦ 0,x,
↦ 0,-1
print(hne_extended[1]);
↦ 0,x, 0,0,0,0, 0,0,0,0,
↦ 0,-1,0,2,0,-4,0,8,0,-16
// ... and the resulting parametrizations:
param(Hne);
↦ // ** Warning: result is exact up to order 2 in x and 3 in y !
↦ _[1]=-t2
↦ _[2]=-t3
param(hne_extended);
↦ // ** Warning: result is exact up to order 10 in x and 11 in y !
↦ _[1]=-t2+2t4-4t6+8t8-16t10
↦ _[2]=-t3+2t5-4t7+8t9-16t11
param(hne_extended,0);
↦ // ** Warning: result is exact up to order 10 in x and 11 in y !
↦ [1]:
↦    _[1]=-t2+2t4-4t6+8t8-16t10
↦    _[2]=-t3+2t5-4t7+8t9-16t11
↦ [2]:
↦    10,11
// An example with more than one branch:
list L=hnexpansion(f*(x2+y4));
↦ 
↦ // 'hnexpansion' created a list of one ring.
↦ // To see the ring and the data stored in the ring, type (if you assigned
↦ // the name L to the list):
↦      show(L);
↦ // To display the computed HN expansion, type
↦      def HNring = L[1]; setring HNring;  displayHNE(hne);
def HNring = L[1]; setring HNring;
param(hne);
```

```
↦ // Parametrization of branch number 1 computed.
↦ // ** Warning: result is exact up to order 4 in x and 5 in y !
↦ // Parametrization of branch number 2 computed.
↦ // Parametrization of branch number 3 computed.
↦ [1]:
↦    _[1]=-x2+2*x4
↦    _[2]=-x3+2*x5
↦ [2]:
↦    _[1]=(a)*x2
↦    _[2]=x
↦ [3]:
↦    _[1]=(-a)*x2
↦    _[2]=x
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.3 [extdevelop], page 1722.

D.6.15.5 displayHNE

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: displayHNE(L[,n]); L list, n int

Assume: L is the output of `develop(f)`, or of `exdevelop(f,n)`, or of `hnexpansion(f[,"ess"])`, or (one entry in) the list `hne` in the ring created by `hnexpansion(f[,"ess"])`.

Return: - if only one argument is given and if the input are the HN data of an irreducible plane curve singularity, no return value, but display an ideal HNE of the following form:

```
y = []*x^1+[]*x^2   +...+x^<>*z(1)
x =        []*z(1)^2+...+z(1)^<>*z(2)
z(1) =     []*z(2)^2+...+z(2)^<>*z(3)
.......             .........................
z(r-1) =   []*z(r)^2+[]*z(r)^3+......
```

where `x`,`y` are the first 2 variables of the basering. The values of `[]` are the coefficients of the Hamburger-Noether matrix, the values of `<>` are represented by `x` in the HN matrix.
- if a second argument is given and if the input are the HN data of an irreducible plane curve singularity, return a ring containing an ideal `HNE` as described above.
- if L corresponds to the output of `hnexpansion(f)` or to the list of HN data computed by `hnexpansion(f[,"ess"])`, `displayHNE(L[,n])` shows the HNE's of all branches of f in the format described above. The optional parameter is then ignored.

Note: The 1st line of the above ideal (i.e., `HNE[1]`) means that `y=[]*z(0)^1+...`, the 2nd line (`HNE[2]`) means that `x=[]*z(1)^2+...`, so you can see which indeterminate corresponds to which line (it's also possible that `x` corresponds to the 1st line and `y` to the 2nd).

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
poly f=x3+2xy2+y2;
list hn=develop(f);
displayHNE(hn);
↦   y = z(1)*x
↦   x = -z(1)^2 + ..... (terms of degree >=3)
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719.

D.6.15.6 invariants

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: invariants(INPUT); INPUT list or poly

Assume: `INPUT` is the output of `develop(f)`, or of `extdevelop(develop(f),n)`, or one entry of the list of HN data computed by `hnexpansion(f[,"ess"])`.

Return: list `INV` of the following format:

INV[1]: intvec (characteristic exponents)
INV[2]: intvec (generators of the semigroup)
INV[3]: intvec (Puiseux pairs, 1st components)
INV[4]: intvec (Puiseux pairs, 2nd components)
INV[5]: int (degree of the conductor)
INV[6]: intvec (sequence of multiplicities)

If `INPUT` contains no valid HN expansion, the empty list is returned.

Assume: `INPUT` is a bivariate polynomial f, or the output of `hnexpansion(f)`, or the list of HN data computed by `hnexpansion(f [,"ess"])`.

Return: list `INV`, such that `INV[i]` coincides with the output of `invariants(develop(f[i]))`, where f[i] is the i-th branch of f, and the last entry of `INV` contains further invariants of f in the format:

INV[last][1] : intmat (contact matrix of the branches)
INV[last][2] : intmat (intersection multiplicities of the branches)
INV[last][3] : int (delta invariant of f)

Note: In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.

Example:

```
LIB "hnoether.lib";
ring exring=0,(x,y),dp;
list Hne=develop(y4+2x3y2+x6+x5y);
list INV=invariants(Hne);
INV[1];                    // the characteristic exponents
↦ 4,6,7
INV[2];                    // the generators of the semigroup of values
↦ 4,6,13
INV[3],INV[4];             // the Puiseux pairs in packed form
↦ 3,7 2,2
INV[5] div 2;              // the delta-invariant
↦ 8
INV[6];                    // the sequence of multiplicities
↦ 4,2,2,1,1
// To display the invariants more 'nicely':
displayInvariants(Hne);
↦  characteristic exponents  : 4,6,7
↦  generators of semigroup   : 4,6,13
↦  Puiseux pairs             : (3,2)(7,2)
↦  degree of the conductor   : 16
↦  delta invariant           : 8
↦  sequence of multiplicities: 4,2,2,1,1
```

```
//////////////////////////////
INV=invariants((x2-y3)*(x3-y5));
INV[1][1];                // the characteristic exponents of the first branch
↦ 2,3
INV[2][6];                // the sequence of multiplicities of the second branch
↦ 3,2,1,1
print(INV[size(INV)][1]);         // the contact matrix of the branches
↦      0     3
↦      3     0
print(INV[size(INV)][2]);         // the intersection numbers of the branches
↦      0     9
↦      9     0
INV[size(INV)][3];                // the delta invariant of the curve
↦ 14
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.7 [displayInvariants], page 1727; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.10 [intersection], page 1730; Section D.6.15.8 [multsequence], page 1727.

D.6.15.7 displayInvariants

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: displayInvariants(INPUT); INPUT list or poly

Assume: `INPUT` is a bivariate polynomial, or the output of `develop(f)`, resp. of `extdevelop(develop(f),n)`, or (one entry of) the list of HN data computed by `hnexpansion(f[,"ess"])`.

Return: none

Display: invariants of the corresponding branch, resp. of all branches, in a better readable form.

Note: If the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.

Example:

```
LIB "hnoether.lib";
ring exring=0,(x,y),dp;
list Hne=develop(y4+2x3y2+x6+x5y);
displayInvariants(Hne);
↦  characteristic exponents  : 4,6,7
↦  generators of semigroup   : 4,6,13
↦  Puiseux pairs             : (3,2)(7,2)
↦  degree of the conductor   : 16
↦  delta invariant           : 8
↦  sequence of multiplicities: 4,2,2,1,1
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.10 [intersection], page 1730; Section D.6.15.6 [invariants], page 1726.

D.6.15.8 multsequence

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: multsequence(INPUT); INPUT list or poly

Assume: INPUT is the output of develop(f), or of extdevelop(develop(f),n), or one entry of the list of HN data computed by hnexpansion(f[,"ess"]).

Return: intvec corresponding to the multiplicity sequence of the irreducible plane curve singularity described by the HN data (return value coincides with invariants(INPUT)[6]).

Assume: INPUT is a bivariate polynomial f, or the output of hnexpansion(f), or the list of HN data computed by hnexpansion(f [,"ess"]).

Return: list of two integer matrices:

multsequence(INPUT)[1][i,*]
: contains the multiplicities of the branches at their infinitely near point of 0 in its (i-1) order neighbourhood (i.e., i=1: multiplicity of the branches themselves, i=2: multiplicity of their 1st quadratic transform, etc.,
Hence, multsequence(INPUT)[1][*,j] is the multiplicity sequence of branch j.

multsequence(INPUT)[2][i,*]:
: contains the information which of these infinitely near points coincide.

Note: The order of the elements of the list of HN data obtained from hnexpansion(f [,"ess"]) must not be changed (because otherwise the coincident infinitely near points couldn't be grouped together, see the meaning of the 2nd intmat in the example).
Hence, it is not wise to compute the HN expansion of polynomial factors separately, put them into a list INPUT and call multsequence(INPUT).
Use displayMultsequence to produce a better readable output for reducible curves on the screen.
In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of hnexpansion and use it as input instead of the polynomial itself.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
list Hne=hnexpansion((x6-y10)*(x+y2-y3)*(x+y2+y3));
↦ // No change of ring necessary, return value is HN expansion.
multsequence(Hne[1]),"  |  ",multsequence(Hne[2]),"  |  ",
multsequence(Hne[3]),"  |  ",multsequence(Hne[4]);
↦ 3,2,1,1   |   3,2,1,1   |   1   |   1
multsequence(Hne);
↦ [1]:
↦    3,3,1,1,
↦    2,2,1,1,
↦    1,1,1,1,
↦    1,1,1,1,
↦    1,1,1,1
↦ [2]:
↦    4,0,0,0,
↦    4,0,0,0,
↦    2,2,0,0,
↦    2,1,1,0,
↦    1,1,1,1
// The meaning of the entries of the 2nd matrix is as follows:
displayMultsequence(Hne);
```

```
↦ [(3,3,1,1)],
↦ [(2,2,1,1)],
↦ [(1,1),(1,1)],
↦ [(1,1),(1),(1)],
↦ [(1),(1),(1),(1)]
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.9 [displayMultsequence], page 1729; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.17 [separateHNE], page 1733.

D.6.15.9 displayMultsequence

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: displayMultsequence(INPUT); INPUT list or poly

Assume: `INPUT` is a bivariate polynomial, or the output of `develop(f)`, resp. of `extdevelop(develop(f),n)`, or (one entry of) the list of HN data computed by `hnexpansion(f[,"ess"])`, or the output of `hnexpansion(f)`.

Return: nothing

Display: the sequence of multiplicities:

- if `INPUT=develop(f)` or `INPUT=extdevelop(develop(f),n)` or `INPUT=hne[i]`:

```
              a , b , c , ....... , 1
```

- if `INPUT=f` or `INPUT=hnexpansion(f)` or `INPUT=hne`:

```
              [(a_1, .... , b_1 , .... , c_1)],
              [(a_2, ... ), ... , (... , c_2)],
               ........................................ ,
              [(a_n),(b_n), ....., (c_n)]
```

with:
`a_1 , ... , a_n` the sequence of multiplicities of the 1st branch,
`[...]` the multiplicities of the j-th transform of all branches,
`(...)` indicating branches meeting in an infinitely near point.

Note: The Same restrictions as in `multsequence` apply for the input.
In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
// Example 1: Input = output of develop
displayMultsequence(develop(x3-y5));
↦ The sequence of multiplicities is   3,2,1,1
// Example 2: Input = bivariate polynomial
displayMultsequence((x6-y10)*(x+y2-y3)*(x+y2+y3));
↦ [(3,3,1,1)],
↦ [(2,2,1,1)],
↦ [(1,1),(1,1)],
↦ [(1,1),(1),(1)],
↦ [(1),(1),(1),(1)]
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.8 [multsequence], page 1727; Section D.6.15.17 [separateHNE], page 1733.

D.6.15.10 intersection

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: intersection(hne1,hne2); hne1, hne2 lists

Assume: `hne1, hne2` represent an HN expansion of an irreducible plane curve singularity (that is, are the output of `develop(f)`, or of `extdevelop(develop(f),n)`, or one entry of the list of HN data computed by `hnexpansion(f[,"ess"])`).

Return: int, the intersection multiplicity of the irreducible plane curve singularities corresponding to `hne1` and `hne2`.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
list Hne=hnexpansion((x2-y3)*(x2+y3));
↦ // No change of ring necessary, return value is HN expansion.
intersection(Hne[1],Hne[2]);
↦ 6
```

See also: Section D.6.15.7 [displayInvariants], page 1727; Section D.6.15.1 [hnexpansion], page 1719.

D.6.15.11 is_irred

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: is_irred(f); f poly

Assume: f is a squarefree bivariate polynomial (in the first 2 ring variables).

Return: int (0 or 1):
- `is_irred(f)=1` if f is irreducible as a formal power series over the algebraic closure of its coefficient field (f defines an analytically irreducible curve at zero),
- `is_irred(f)=0` otherwise.

Note: 0 and units in the ring of formal power series are considered to be not irreducible.

Example:

```
LIB "hnoether.lib";
ring exring=0,(x,y),ls;
is_irred(x2+y3);
↦ 1
is_irred(x2+y2);
↦ 0
is_irred(x2+y3+1);
↦ 0
```

D.6.15.12 delta

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: delta(INPUT); INPUT a polynomial defining an isolated plane curve singularity at 0, or the Hamburger-Noether expansion thereof, i.e. the output of `develop(f)`, or the output of `hnexpansion(f)`, or the list of HN data computed by `hnexpansion(f)`.

Return: int, the delta invariant of the singularity at 0, that is, the vector space dimension of R˜/R, (R˜ the normalization of the local ring of the singularity).

Note: In case the Hamburger-Noether expansion of the curve f is needed for other purposes as well it is better to calculate this first with the aid of `hnexpansion` and use it as input instead of the polynomial itself.

Example:

```
LIB "hnoether.lib";
ring r = 32003,(x,y),ds;
poly f = x25+x24-4x23-1x22y+4x22+8x21y-2x21-12x20y-4x19y2+4x20+10x19y
+12x18y2-24x18y-20x17y2-4x16y3+x18+60x16y2+20x15y3-9x16y
-80x14y3-10x13y4+36x14y2+60x12y4+2x11y5-84x12y3-24x10y5
+126x10y4+4x8y6-126x8y5+84x6y6-36x4y7+9x2y8-1y9;
delta(f);
↦ 96
```

See also: Section D.4.23.11 [deltaLoc], page 1203; Section D.6.15.6 [invariants], page 1726.

D.6.15.13 newtonpoly

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: newtonpoly(f); f poly

Assume: basering has exactly two variables;
f is convenient, that is, f(x,0) != 0 != f(0,y).

Return: list of intvecs (= coordinates x,y of the Newton polygon of f).

Note: Procedure uses `execute`; this can be avoided by calling `newtonpoly(f,1)` if the ordering of the basering is `ls`.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),ls;
poly f=x5+2x3y-x2y2+3xy5+y6-y7;
newtonpoly(f);
↦ [1]:
↦    0,6
↦ [2]:
↦    2,2
↦ [3]:
↦    3,1
↦ [4]:
↦    5,0
```

D.6.15.14 is_NND

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: is_NND(f[,mu,NP]); f poly, mu int, NP list of intvecs

Assume: f is convenient, that is, f(x,0) != 0 != f(0,y);
mu (optional) is Milnor number of f.
NP (optional) is output of `newtonpoly(f)`.

Return: int: 1 if f is Newton non-degenerate, 0 otherwise.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),ls;
poly f=x5+y3;
is_NND(f);
↦ 1
poly g=(x-y)^5+3xy5+y6-y7;
is_NND(g);
↦ 0
// if already computed, one should give the Minor number and Newton polygon
// as second and third input:
int mu=milnor(g);
list NP=newtonpoly(g);
is_NND(g,mu,NP);
↦ 0
```

See also: Section D.6.15.13 [newtonpoly], page 1731.

D.6.15.15 stripHNE

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: stripHNE(L); L list

Assume: L is the output of `develop(f)`, or of `extdevelop(develop(f),n)`, or (one entry of) the list `hne` in the ring created by `hnexpansion(f[,"ess"])`.

Return: list in the same format as L, but all polynomials L[4], resp. L[i][4], are set to zero.

Note: The purpose of this procedure is to remove huge amounts of data no longer needed. It is useful, if one or more of the polynomials in L consume much memory. It is still possible to compute invariants, parametrizations etc. with the stripped HNE(s), but it is not possible to use `extdevelop` with them.

Example:

```
LIB "hnoether.lib";
ring r=0,(x,y),dp;
list Hne=develop(x2+y3+y4);
Hne;
↦ [1]:
↦    _[1,1]=0
↦    _[1,2]=x
↦    _[2,1]=0
↦    _[2,2]=-1
↦ [2]:
↦    1,-1
↦ [3]:
↦    1
↦ [4]:
↦    x4-2x2y+y2+y
↦ [5]:
↦    1
stripHNE(Hne);
↦ [1]:
↦    _[1,1]=0
↦    _[1,2]=x
↦    _[2,1]=0
↦    _[2,2]=-1
↦ [2]:
↦    1,-1
↦ [3]:
↦    1
↦ [4]:
↦    0
↦ [5]:
↦    1
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.3 [extdevelop], page 1722; Section D.6.15.1 [hnexpansion], page 1719.

D.6.15.16 puiseux2generators

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: puiseux2generators(m,n); m,n intvec

Assume: m, resp. n, represent the 1st, resp. 2nd, components of Puiseux pairs (e.g., `m=invariants(L)[3]`, `n=invariants(L)[4]`).

Return: intvec of the generators of the semigroup of values.

Example:

```
LIB "hnoether.lib";
// take (3,2),(7,2),(15,2),(31,2),(63,2),(127,2) as Puiseux pairs:
puiseux2generators(intvec(3,7,15,31,63,127),intvec(2,2,2,2,2,2));
↦ 64,96,208,424,852,1706,3413
```

See also: Section D.6.15.6 [invariants], page 1726.

D.6.15.17 separateHNE

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: separateHNE(hne1,hne2); hne1, hne2 lists

Assume: hne1, hne2 are HNEs (=output of `develop(f)`, `extdevelop(develop(f),n)`, or one entry in the list `hne` in the ring created by `hnexpansion(f[,"ess"])`.

Return: number of quadratic transformations needed to separate both curves (branches).

Example:

```
LIB "hnoether.lib";
int p=printlevel; printlevel=-1;
ring r=0,(x,y),dp;
list hne1=develop(x);
list hne2=develop(x+y);
list hne3=develop(x+y2);
separateHNE(hne1,hne2);  // two transversal lines
↦ 1
separateHNE(hne1,hne3);  // one quadratic transform. gives 1st example
↦ 2
printlevel=p;
```

See also: Section D.6.15.2 [develop], page 1721; Section D.6.15.9 [displayMultsequence], page 1729; Section D.6.15.1 [hnexpansion], page 1719; Section D.6.15.8 [multsequence], page 1727.

D.6.15.18 squarefree

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: squarefree(f); f poly

Assume: f is a bivariate polynomial (in the first 2 ring variables).

Return: poly, a squarefree divisor of f.

Note: Usually, the return value is the greatest squarefree divisor, but there is one exception: factors with a p-th root, p the characteristic of the basering, are lost.

Example:

```
LIB "hnoether.lib";
ring exring=3,(x,y),dp;
squarefree((x3+y)^2);
↦ x3+y
squarefree((x+y)^3*(x-y)^2); // Warning: (x+y)^3 is lost
↦ x-y
squarefree((x+y)^4*(x-y)^2); // result is (x+y)*(x-y)
↦ x2-y2
```

See also: Section D.6.15.19 [allsquarefree], page 1734.

D.6.15.19 allsquarefree

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage : allsquarefree(f,g); f,g poly

Assume: g is the output of `squarefree(f)`.

Return: the greatest squarefree divisor of f.

Note : This proc uses factorize to get the missing factors of f not in g and, therefore, may be slow.

Example:

```
LIB "hnoether.lib";
ring exring=7,(x,y),dp;
poly f=(x+y)^7*(x-y)^8;
poly g=squarefree(f);
g;                       // factor x+y lost, since characteristic=7
↦ x-y
allsquarefree(f,g);      // all factors (x+y)*(x-y) found
↦ x2-y2
```

See also: Section D.6.15.18 [squarefree], page 1734.

D.6.15.20 further_hn_proc

Procedure from library `hnoether.lib` (see Section D.6.15 [hnoether_lib], page 1718).

Usage: further_hn_proc();

Note: The library `hnoether.lib` contains some more procedures which are not shown when typing `help hnoether.lib;`. They may be useful for interactive use (e.g. if you want to do the calculation of an HN development "by hand" to see the intermediate results), and they can be enumerated by calling `further_hn_proc()`.
Use `help <procedure>;` for detailed information about each of them.

Example:

```
LIB "hnoether.lib";
further_hn_proc();
↦
↦  The following procedures are also part of 'hnoether.lib':
↦
↦  getnm(f);           intersection pts. of Newton polygon with axes
↦  T_Transform(f,Q,N); returns f(y,xy^Q)/y^NQ (f: poly, Q,N: int)
↦  T1_Transform(f,d,M); returns f(x,y+d*x^M)  (f: poly,d:number,M:int)
↦  T2_Transform(f,d,M,N,ref);   a composition of T1 & T
↦  koeff(f,I,J);       gets coefficient of indicated monomial of polynomial\
   f
↦  redleit(f,S,E);     restriction of monomials of f to line (S-E)
↦  leit(f,n,m);        special case of redleit (for irred. polynomials)
↦  testreducible(f,n,m); tests whether f is reducible
↦  charPoly(f,M,N);    characteristic polynomial of f
↦  find_in_list(L,p);  find int p in list L
↦  get_last_divisor(M,N); last divisor in Euclid's algorithm
↦  factorfirst(f,M,N); try to factor f without 'factorize'
↦  factorlist(L);      factorize a list L of polynomials
↦  referencepoly(D);   a polynomial f s.t. D is the Newton diagram of f
```

D.6.16 kskernel_lib

Library: kskernel.lib

Purpose: procedures for computing the kernel of the kodaira-spencer map

Author: Tetyana Povalyaeva, povalyae@mathematik.uni-kl.de

Procedures:

D.6.16.1 KSker

Procedure from library `kskernel.lib` (see Section D.6.16 [kskernel_lib], page 1735).

Usage: KSker(int p,q); p,q relatively prime integers

Return: nothing; exports ring KSring, matrix KSkernel and list 'weights'; KSkernel is a matrix of coefficients of the
generators of the kernel of Kodaira-Spencer map,
'weights' is a list of degrees for variables T

Example:

```
LIB "kskernel.lib";
int p=6;
int q=7;
KSker(p,q);
setring Kskernel::KSring;
print(KSkernel);
↦ 0,      0,      0,             0,             0,             2*T(1),
↦ 0,      0,      0,             0,             0,             3*T(2),
↦ 0,      0,      0,             0,             0,             4*T(3),
↦ 0,      0,      0,             2*T(1),        3*T(2),        9*T(4),
↦ 0,      0,      0,             3*T(2),        KSkernel[5,5],10*T(5),
↦ 2*T(1),3*T(2),KSkernel[6,3],KSkernel[6,4],KSkernel[6,5],16*T(6)
```

D.6.16.2 KSconvert

Procedure from library `kskernel.lib` (see Section D.6.16 [kskernel_lib], page 1735).

Usage: KSconvert(matrix M);
M is a matrix of coefficients of the generators of
the kernel of Kodaira-Spencer map in variables T(i)
from the basering. To be called after the procedure
KSker(p,q)

Return: nothing; exports ring KSring2 and matrix KSkernel2 within it, such that KSring2 resp. KSkernel2 are in variables
T(w) with weights -w. These weights are computed
in the procedure KSker(p,q)

Example:

```
LIB "kskernel.lib";
int p=6;
int q=7;
KSker(p,q);
setring Kskernel::KSring;
KSconvert(KSkernel);
↦ // ** `KSring2` is already global
setring Kskernel::KSring2;
print(KSkernel2);
↦ 0,     0,     0,             0,             0,             2*T(2),
↦ 0,     0,     0,             0,             0,             3*T(3),
↦ 0,     0,     0,             0,             0,             4*T(4),
↦ 0,     0,     0,             2*T(2),        3*T(3),        9*T(9),
↦ 0,     0,     0,             3*T(3),        KSkernel2[5,5],10*T(10),
↦ 2*T(2),3*T(3),KSkernel2[6,3],KSkernel2[6,4],KSkernel2[6,5],16*T(16)
```

D.6.16.3 KSlinear

Procedure from library `kskernel.lib` (see Section D.6.16 [kskernel_lib], page 1735).

Usage: KSlinear(matrix M);
computes matrix of linear terms of the kernel of the
Kodaira-Spencer map. To be called after the procedure
KSker(p,q)

Return: nothing; but replaces elements of the matrix KSkernel in the ring Ksring with their leading monomials
w.r.t. the local ordering (ls)

Example:

```
LIB "kskernel.lib";
int p=6;
int q=7;
KSker(p,q);
setring Kskernel::KSring;
KSlinear(KSkernel);
print(KSkernel);
↦ 0,     0,     0,     0,     0,      2*T(1),
↦ 0,     0,     0,     0,     0,      3*T(2),
↦ 0,     0,     0,     0,     0,      4*T(3),
↦ 0,     0,     0,     2*T(1),3*T(2), 9*T(4),
↦ 0,     0,     0,     3*T(2),4*T(3), 10*T(5),
↦ 2*T(1),3*T(2),4*T(3),9*T(4),10*T(5),16*T(6)
```

D.6.16.4 KScoef

Procedure from library kskernel.lib (see Section D.6.16 [kskernel_lib], page 1735).

Return: exports ring RC and number C within it. C is
the coefficient of the word defined in the list qq,
being a part of C[i,j] for x^p+y^q

Example:

```
LIB "kskernel.lib";
int p=5; int q=14;
int i=2; int j=9;
list L;
ring r=0,x,dp;
number c;
L[1]=3; L[2]=1; L[3]=3; L[4]=2;
KScoef(i,j,p,q,L);
c=imap(RC,C);
c;
↦ 27/8575
L[1]=3; L[2]=1; L[3]=2; L[4]=3;
KScoef(i,j,p,q,L);
↦ // ** redefining S (  export S;)
↦ // ** redefining u (  export u;)
↦ // ** redefining l (  export l;)
↦ // ** redefining RC (  export RC;exportto(Top,RC);)
c=c+imap(RC,C);
c; // it is a coefficient of T1*T2*T3^2 in C[2,9] for x^5+y^14
↦ 99/8575
```

D.6.16.5 StringF

Procedure from library kskernel.lib (see Section D.6.16 [kskernel_lib], page 1735).

Usage: StringF(int i,j,p,q);

Return: nothing; exports string F which contains an expression in variables T(i) with non-resolved brackets

Example:

```
LIB "kskernel.lib";
int p=5; int q=14;
int i=2; int j=9;
StringF(i,j,p,q);
F;
↦ T(7)+T(3)*(T(4)*(T(1))+T(1)*(T(4)+T(3)*(T(2))+T(2)*(T(3)+T(1)*(T(1)))))
```

D.6.17 mondromy_lib

Library: mondromy.lib

Purpose: Monodromy of an Isolated Hypersurface Singularity

Author: Mathias Schulze, email: mschulze@mathematik.uni-kl.de

Overview: A library to compute the monodromy of an isolated hypersurface singularity. It uses an algorithm by Brieskorn (manuscripta math. 2 (1970), 103-161) to compute a connection matrix of the meromorphic Gauss-Manin connection up to arbitrarily high order, and an algorithm of Gerard and Levelt (Ann. Inst. Fourier, Grenoble 23,1 (1973), pp. 157-195) to transform it to a simple pole.

Procedures: See also: Section D.6.14 [gmspoly_lib], page 1717; Section D.6.13 [gmssing_lib], page 1702.

D.6.17.1 detadj

Procedure from library `mondromy.lib` (see Section D.6.17 [mondromy_lib], page 1737).

Usage: detadj(U); U matrix

Assume: U is a square matrix with non zero determinant.

Return: The procedure returns a list with at most 2 entries.
If U is not a square matrix, the list is empty.
If U is a square matrix, then the first entry is the determinant of U. If U is a square matrix and the determinant of U not zero, then the second entry is the adjoint matrix of U.

Display: The procedure displays comments if printlevel>=1.

Example:

```
LIB "mondromy.lib";
ring R=0,x,dp;
matrix U[2][2]=1,1+x,1+x2,1+x3;
list daU=detadj(U);
daU[1];
↦ -x2-x
print(daU[2]);
↦ x3+1, -x-1,
↦ -x2-1,1
```

D.6.17.2 invunit

Procedure from library `mondromy.lib` (see Section D.6.17 [mondromy_lib], page 1737).

Usage: invunit(u,n); u poly, n int

Assume: The polynomial u is a series unit.

Return: The procedure returns the series inverse of u up to order n or a zero polynomial if u is no series unit.

Display: The procedure displays comments if printlevel>=1.

Example:

```
LIB "mondromy.lib";
ring R=0,(x,y),dp;
invunit(2+x3+xy4,10);
↦ 1/8x2y8-1/16x9+1/4x4y4+1/8x6-1/4xy4-1/4x3+1/2
```

D.6.17.3 jacoblift

Procedure from library `mondromy.lib` (see Section D.6.17 [mondromy_lib], page 1737).

Usage: jacoblift(f); f poly

Assume: The polynomial f in a series ring (local ordering) defines an isolated hypersurface singularity.

Return: The procedure returns a list with entries kappa, xi, u of type int, vector, poly such that kappa is minimal with f^kappa in jacob(f), u is a unit, and u*f^kappa=(matrix(jacob(f))*xi)[1,1].

Display: The procedure displays comments if printlevel>=1.

Example:

```
LIB "mondromy.lib";
ring R=0,(x,y),ds;
poly f=x2y2+x6+y6;
jacoblift(f);
↦ [1]:
↦    2
↦ [2]:
↦    1/2x2y3*gen(2)+1/6x7*gen(1)+5/6x6y*gen(2)-2/3xy6*gen(1)+1/6y7*gen(2)-4\
  x4y5*gen(2)-3/2x9y2*gen(1)-15/2x8y3*gen(2)+9/2x3y8*gen(1)-3/2x2y9*gen(2)
↦ [3]:
↦    1-9x2y2
```

D.6.17.4 monodromyB

Procedure from library `mondromy.lib` (see Section D.6.17 [mondromy_lib], page 1737).

Usage: monodromyB(f[,opt]); f poly, opt int

Assume: The polynomial f in a series ring (local ordering) defines an isolated hypersurface singularity.

Return: The procedure returns a residue matrix M of the meromorphic Gauss-Manin connection of the singularity defined by f or an empty matrix if the assumptions are not fulfilled. If opt=0 (default), exp(-2*pi*i*M) is a monodromy matrix of f, else, only the characteristic polynomial of exp(-2*pi*i*M) coincides with the characteristic polynomial of the monodromy of f.

Display: The procedure displays more comments for higher printlevel.

Example:

```
LIB "mondromy.lib";
ring R=0,(x,y),ds;
poly f=x2y2+x6+y6;
matrix M=monodromyB(f);
print(M);
↦ 7/6,0,  0,0,  0,  0,0,   0,-1/2,0,  0,  0,  0,
↦ 0,  7/6,0,0,  0,  0,-1/2,0,0,   0,  0,  0,  0,
↦ 0,  0,  1,0,  0,  0,0,   0,0,   0,  0,  0,  0,
↦ 0,  0,  0,4/3,0,  0,0,   0,0,   0,  0,  0,  0,
↦ 0,  0,  0,0,  4/3,0,0,   0,0,   0,  0,  0,  0,
↦ 0,  0,  0,0,  0,  1,0,   0,0,   0,  0,  0,  0,
↦ 0,  0,  0,0,  0,  0,5/6, 0,0,   0,  0,  0,  0,
↦ 0,  0,  0,0,  0,  0,0,   1,0,   0,  0,  0,  0,
↦ 0,  0,  0,0,  0,  0,0,   0,5/6, 0,  0,  0,  0,
↦ 0,  0,  0,0,  0,  0,0,   0,0,   2/3,0,  0,  0,
↦ 0,  0,  0,0,  0,  0,0,   0,0,   0,  2/3,0,  0,
↦ 0,  0,  0,0,  0,  0,0,   0,0,   0,  0,  1,  -1/3,
↦ 0,  0,  0,0,  0,  0,0,   0,0,   0,  0,  3/4,0
```

D.6.17.5 H2basis

Procedure from library `mondromy.lib` (see Section D.6.17 [mondromy_lib], page 1737).

Usage: H2basis(f); f poly

Assume: The polynomial f in a series ring (local ordering) defines an isolated hypersurface singularity.

Return: The procedure returns a list of representatives of a C{f}-basis of the Brieskorn lattice H"=Omega^(n+1)/df^dOmega^(n-1).

Theory: H" is a free C{f}-module of rank milnor(f).

Display: The procedure displays more comments for higher printlevel.

Example:

```
LIB "mondromy.lib";
ring R=0,(x,y),ds;
poly f=x2y2+x6+y6;
H2basis(f);
↦ [1]:
↦    x4
↦ [2]:
↦    x2y2
↦ [3]:
↦    y4
↦ [4]:
↦    x3
↦ [5]:
↦    x2y
↦ [6]:
↦    xy2
↦ [7]:
↦    y3
↦ [8]:
↦    x2
↦ [9]:
↦    xy
↦ [10]:
↦    y2
↦ [11]:
↦    x
↦ [12]:
↦    y
↦ [13]:
↦    1
```

D.6.18 qhmoduli_lib

Library: qhmoduli.lib

Purpose: Moduli Spaces of Semi-Quasihomogeneous Singularities

Author: Thomas Bayer, email: bayert@in.tum.de

Procedures:

D.6.18.1 ArnoldAction

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: ArnoldAction(f, [Gf, B]); poly f; list Gf, B;
'Gf' is a list of two rings (coming from 'StabEqn')

Purpose: compute the induced action of the stabilizer G of f on T_, where T_ is given by the upper monomials B of the Milnor algebra of f.

Assume: f is quasihomogeneous

Return: polynomial ring over the same ground field, containing the ideals 'actionid' and 'stabid'.
- 'actionid' is the ideal defining the induced action of Gf on T_
- 'stabid' is the ideal of the stabilizer Gf in the new ring

Example:

```
LIB "qhmoduli.lib";
ring B   = 0,(x,y,z), ls;
poly f = -z5+y5+x2z+x2y;
def R = ArnoldAction(f);
↦ // ** redefining zz (int zz = 1;) qhmoduli.lib::StabEqnId:629
setring R;
actionid;
↦ actionid[1]=-s(2)*t(1)+s(3)*t(1)
↦ actionid[2]=-s(2)^2*t(2)+2*s(2)^2*t(3)^2+s(3)^2*t(2)
↦ actionid[3]=s(2)*t(3)+s(3)*t(3)
stabid;
↦ stabid[1]=s(2)*s(3)
↦ stabid[2]=s(1)^2*s(2)+s(1)^2*s(3)-1
↦ stabid[3]=s(1)^2*s(3)^2-s(3)
↦ stabid[4]=s(1)^2+s(2)^4-s(3)^4
↦ stabid[5]=s(1)^4+s(2)^3-s(3)^3
↦ stabid[6]=-s(1)^2*s(3)+s(3)^5
```

D.6.18.2 ModEqn

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: ModEqn(f [, opt]); poly f; int opt;

Purpose: compute equations of the moduli space of semiquasihomogenos hypersurface singularity with principal part f w.r.t. right equivalence

Assume: f quasihomogeneous polynomial with an isolated singularity at 0

Return: polynomial ring, possibly a simple extension of the ground field of the basering, containing the ideal 'modid'
- 'modid' is the ideal of the moduli space if opt is even (> 0). otherwise it contains generators of the coordinate ring R of the moduli space (note : Spec(R) is the moduli space)

Options: 1 compute equations of the mod. space,
2 use a primary decomposition,
4 compute E_f0, i.e., the image of G_f0,
to combine options, add their value, default: opt =7

Example:

```
LIB "qhmoduli.lib";
ring B   = 0,(x,y), ls;
poly f = -x4 + xy5;
def R = ModEqn(f);
↦ // ** redefining zz (int zz = 1;) qhmoduli.lib::StabEqnId:629
setring R;
modid;
↦ modid[1]=Y(5)^2-Y(4)*Y(6)
↦ modid[2]=Y(4)*Y(5)-Y(3)*Y(6)
↦ modid[3]=Y(3)*Y(5)-Y(2)*Y(6)
↦ modid[4]=Y(2)*Y(5)-Y(1)*Y(6)
↦ modid[5]=Y(4)^2-Y(3)*Y(5)
↦ modid[6]=Y(3)*Y(4)-Y(2)*Y(5)
↦ modid[7]=Y(2)*Y(4)-Y(1)*Y(5)
↦ modid[8]=Y(3)^2-Y(2)*Y(4)
↦ modid[9]=Y(2)*Y(3)-Y(1)*Y(4)
↦ modid[10]=Y(2)^2-Y(1)*Y(3)
```

D.6.18.3 QuotientEquations

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: QuotientEquations(G,action,emb [, opt]); ideal G,action,emb;int opt

Purpose: compute the quotient of the variety given by the parameterization 'emb' by the linear action 'action' of the algebraic group G.

Assume: 'action' is linear, G must be finite if the Reynolds operator is needed (i.e., NullCone(G,action) returns some non-invariant polys)

Return: polynomial ring over a simple extension of the ground field of the basering, containing the ideals 'id' and 'embedid'.
- 'id' contains the equations of the quotient, if opt = 1; if opt = 0, 2, 'id' contains generators of the coordinate ring R of the quotient (Spec(R) is the quotient)
- 'embedid' = 0, if opt = 1;
if opt = 0, 2, it is the ideal defining the equivariant embedding

Options: 1 compute equations of the quotient,
2 use a primary decomposition when computing the Reynolds operator,
to combine options, add their value, default: opt =3.

D.6.18.4 StabEqn

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: StabEqn(f); f polynomial

Purpose: compute the equations of the isometry group of f.

Assume: f semiquasihomogeneous polynomial with an isolated singularity at 0

Return: list of two rings 'S1', 'S2'
- 'S1' contains the equations of the stabilizer (ideal 'stabid')
- 'S2' contains the action of the stabilizer (ideal 'actionid')

Global: varSubsList, contains the index j s.t. x(i) -> x(i)t(j) ...

Example:

```
LIB "qhmoduli.lib";
ring B = 0,(x,y,z), ls;
poly f = -z5+y5+x2z+x2y;
list stab = StabEqn(f);
↦ // ** redefining zz (int zz = 1;) qhmoduli.lib::StabEqnId:629
def S1 = stab[1]; setring S1;  stabid;
↦ stabid[1]=s(2)*s(3)
↦ stabid[2]=s(1)^2*s(2)+s(1)^2*s(3)-1
↦ stabid[3]=s(1)^2*s(3)^2-s(3)
↦ stabid[4]=s(2)^4-s(3)^4+s(1)^2
↦ stabid[5]=s(1)^4+s(2)^3-s(3)^3
↦ stabid[6]=s(3)^5-s(1)^2*s(3)
def S2 = stab[2]; setring S2;  actionid;
↦ actionid[1]=s(1)*x
↦ actionid[2]=s(3)*y+s(2)*z
↦ actionid[3]=s(2)*y+s(3)*z
```

D.6.18.5 StabEqnId

Procedure from library qhmoduli.lib (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: StabEqn(I, w); I ideal, w intvec

Purpose: compute the equations of the isometry group of the ideal I, each generator of I is fixed by the stabilizer.

Assume: I semiquasihomogeneous ideal w.r.t. 'w' with an isolated singularity at 0

Return: list of two rings 'S1', 'S2'
- 'S1' contians the equations of the stabilizer (ideal 'stabid')
- 'S2' contains the action of the stabilizer (ideal 'actionid')

Global: varSubsList, contains the index j s.t. t(i) -> t(i)t(j) ...

Example:

```
LIB "qhmoduli.lib";
ring B   = 0,(x,y,z), ls;
ideal I = x2,y3,z6;
intvec w = 3,2,1;
list stab = StabEqnId(I, w);
↦ // ** redefining zz (int zz = 1;) qhmoduli.lib::StabEqnId:629
def S1 = stab[1]; setring S1;  stabid;
↦ stabid[1]=s(1)^2-1
↦ stabid[2]=s(2)^3-1
↦ stabid[3]=s(3)^6-1
```

```
def S2 = stab[2]; setring S2;  actionid;
↦ actionid[1]=s(1)*x
↦ actionid[2]=s(2)*y
↦ actionid[3]=s(3)*z
```

D.6.18.6 StabOrder

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: StabOrder(f); poly f

Purpose: compute the order of the stabilizer group of f.

Assume: f quasihomogeneous polynomial with an isolated singularity at 0

Return: int

Global: varSubsList

D.6.18.7 UpperMonomials

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: UpperMonomials(poly f, [intvec w])

Purpose: compute the upper monomials of the milnor algebra of f.

Assume: f is quasihomogeneous (w.r.t. w)

Return: ideal

Example:

```
LIB "qhmoduli.lib";
ring B   = 0,(x,y,z), ls;
poly f = -z5+y5+x2z+x2y;
UpperMonomials(f);
↦ _[1]=y3z3
↦ _[2]=x2y3
↦ _[3]=x2y2
```

D.6.18.8 Max

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: Max(data); intvec/list of integers

Purpose: find the maximal integer contained in 'data'

Return: list

Assume: 'data' contains only integers and is not empty

Example:

```
LIB "qhmoduli.lib";
Max(list(1,2,3));
↦ 3
```

D.6.18.9 Min

Procedure from library `qhmoduli.lib` (see Section D.6.18 [qhmoduli_lib], page 1741).

Usage: Min(data); intvec/list of integers

Purpose: find the minimal integer contained in 'data'

Return: list

Assume: 'data' contains only integers and is not empty

Example:

```
LIB "qhmoduli.lib";
Min(intvec(1,2,3));
↦ 1
```

D.6.19 realclassify_lib

Library: realclassify.lib

Purpose: Classification of real singularities

Author: Janko Boehm, boehm@mathematik.uni-kl.de
Magdaleen Marais, magdaleen@aims.ac.za
Andreas Steenpass, steenpass@mathematik.uni-kl.de

Overview: A library for classifying isolated hypersurface singularities over the reals w.r.t. right equivalence, based on the determinator of singularities by V.I. Arnold. This library is based on classify2.lib by the first and second author and G. Pfister, but handles the real case, while classify2.lib does the complex classification.

References:

Arnold, Varchenko, Gusein-Zade: Singularities of Differentiable Maps. Vol. 1: The classification of critical points caustics and wave fronts. Birkh"auser, Boston 1985

J. Boehm, M.S. Marais, A. Steenpass: The Classification of Real Singularities Using Singular. Part III: Unimodal Singularities of Corank 2, https://arxiv.org/abs/1512.09028

Greuel, Lossen, Shustin: Introduction to singularities and deformations. Springer, Berlin 2007

M.S. Marais, A. Steenpass: The Classification of Real Singularities Using SINGULAR. Part I: Splitting Lemma and Simple Singularities, J. Symb. Comput. 68 (2015), 61-71

M.S. Marais, A. Steenpass: The Classification of Real Singularities Using SINGULAR. Part II: The Structure of the Equivalence Classes of the Unimodal Singularities, J. Symb. Comput. 74 (2016), 346-366

Acknowledgements: This research was supported by
the Staff Exchange Bursary Programme of the University of Pretoria, DFG SPP 1489, and DFG TRR 195. The financial assistance of the National Research Foundation (NRF), South Africa, towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the National Research Foundation, South Africa.

Procedures:

D.6.19.1 realclassify

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Usage: realclassify(f); f poly

Return: A list of elements NF of type NormalFormEquation with the following keys:
NF.singularityType = the type of the singularity as a string
NF.normalFormEquation = an element F of type Poly with the normal form equation (a representative of the stable equivalence class of f, as chosen by Arnold)
NF.modality = the modality of f
NF.parameters = a list where each entry is a list consisting out of the parameter term in the normalform equation and an interval
NF.corank = the corank of f (of type int)
NF.inertiaIndex = the interia index of f (of type int)
NF.milnorNumber = the milnor number of f (of type int)
NF.determinacy = an lower bound for the determinacy of f (of type int)

Note: The classification is done over the real numbers, so in contrast to classify.lib, the signs of coefficients of monomials where even exponents occur matter.
The ground field must be Q (the rational numbers). No field extensions of any kind nor floating point numbers are allowed.
The monomial order must be local.
The input polynomial must be contained in maxideal(2) and must be an isolated singularity of modality 0 or 1. The Milnor number is checked for being finite.
In case of NF.modality=1, the parameter values are given as an element of a number field QQ[a]/(minpoly), the coefficient ring of F in terms of the unique root of minpoly of the ring F.in in the interval specified in NF.parameters in conjunction with the parameter term.

Example:

```
LIB "realclassify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = D[4]-
↦ Normalform equation = x2y-y3
↦ Milnor number = 4
↦ Modality = 0
↦ Determinacy <= 3
↦
map phi = r, x+2y+y^2+x*y,x+y+y^2+x^2,z;
poly f  = x^2+y^2-z^2;  // A[1]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 0
↦ Inertia index = 1
↦ Normalform equation of type = A[1]
↦ Normalform equation = 0
↦ Milnor number = 1
↦ Modality = 0
↦ Determinacy <= 2
↦
poly f  = x^3+y^2-z^2;  // A[2]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 1
↦ Inertia index = 1
↦ Normalform equation of type = A[2]
↦ Normalform equation = x3
↦ Milnor number = 2
↦ Modality = 0
↦ Determinacy <= 3
↦
poly f  = x^2*y-y^3+z^2;  // D[4]-
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = D[4]-
↦ Normalform equation = x2y-y3
↦ Milnor number = 4
↦ Modality = 0
↦ Determinacy <= 3
↦
poly f  = x^3-y^4-z^2;  // E[6]-
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = E[6]-
↦ Normalform equation = x3-y4
↦ Milnor number = 6
↦ Modality = 0
↦ Determinacy <= 4
↦
poly f  = x^3+x*y^3+z^2;  // E[7]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = E[7]
↦ Normalform equation = x3+xy3
↦ Milnor number = 7
↦ Modality = 0
↦ Determinacy <= 5
↦
poly f  = x^3+y^5-z^2;  // E[8]
f=phi(f);
```

```
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = E[8]
↦ Normalform equation = x3+y5
↦ Milnor number = 8
↦ Modality = 0
↦ Determinacy <= 5
↦
poly f  = x^3+3*x^2*y^2+x*y^4-z^2;  //J[10]+
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = J[10]-
↦ Normalform equation = x3+(a)*x2y2-xy4
↦ Milnor number = 10
↦ Modality = 1
↦ Parameter term = (a)*x2y2
↦ Minimal polynomial = (5a4-81)
↦ Interval = [0, 731]
↦ Determinacy <= 7
↦
↦ [2]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = J[10]+
↦ Normalform equation = x3+(a)*x2y2+xy4
↦ Milnor number = 10
↦ Modality = 1
↦ Parameter term = (a)*x2y2
↦ Minimal polynomial = (5a4-81)
↦ Interval = [-5/2, -3/2]
↦ Determinacy <= 7
↦
↦ [3]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = J[10]+
↦ Normalform equation = x3+3x2y2+xy4
↦ Milnor number = 10
↦ Modality = 1
↦ Parameter term = 3x2y2
↦ Determinacy <= 7
↦
poly f  = x^3+x^2*y^2+4*y^9+z^2;  //J[13]+
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = J[13]+
↦ Normalform equation = x3+x2y2+15y9
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = 15y9
↦ Determinacy <= 10
↦ 
↦ [2]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = J[13]+
↦ Normalform equation = x3+x2y2-15y9
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = -15y9
↦ Determinacy <= 10
↦ 
poly f  = -x^4-x^2*y^2+3y^9-z^2;  //X[14]--
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = X[14]--
↦ Normalform equation = -x4-x2y2+19683y9
↦ Milnor number = 14
↦ Modality = 1
↦ Parameter term = 19683y9
↦ Determinacy <= 9
↦ 
↦ [2]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = X[14]--
↦ Normalform equation = -x4-x2y2-19683y9
↦ Milnor number = 14
↦ Modality = 1
↦ Parameter term = -19683y9
↦ Determinacy <= 9
↦ 
poly f  = -x^2*y^2-x^7+4*y^8+z^2;  //Y[7,8]--
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = Y[8,7]-+
↦ Normalform equation = -x2y2+(a)*y7+x8
↦ Milnor number = 16
↦ Modality = 1
↦ Parameter term = (a)*y7
↦ Minimal polynomial = (a4-128)
↦ Interval = [-130, 0]
↦ Determinacy <= 8
↦
↦ [2]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = Y[8,7]-+
↦ Normalform equation = -x2y2+y7+x8
↦ Milnor number = 16
↦ Modality = 1
↦ Parameter term = (a)*y7
↦ Minimal polynomial = (a4-128)
↦ Interval = [0, 130]
↦ Determinacy <= 8
↦
↦ [3]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = Y[7,8]--
↦ Normalform equation = -x2y2-x7+4y8
↦ Milnor number = 16
↦ Modality = 1
↦ Parameter term = 4y8
↦ Determinacy <= 8
↦
↦ [4]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = Y[7,8]-+
↦ Normalform equation = -x2y2+x7+4y8
↦ Milnor number = 16
↦ Modality = 1
↦ Parameter term = 4y8
↦ Determinacy <= 8
↦
poly f  = (x^2+y^2)^2+5*x^9-z^2;  // tilde Y[9]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = tilde(Y)[9]+
↦ Normalform equation = x4+2x2y2+y4+5x9
↦ Milnor number = 19
↦ Modality = 1
↦ Parameter term = 5x9
↦ Determinacy <= 9
↦
↦ [2]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = tilde(Y)[9]+
↦ Normalform equation = x4+2x2y2+y4-5x9
↦ Milnor number = 19
↦ Modality = 1
↦ Parameter term = -5x9
↦ Determinacy <= 9
↦
poly f  = x^3+y^7+3*x*y^5+z^2;  // E[12]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = E[12]
↦ Normalform equation = x3+3xy5+y7
↦ Milnor number = 12
↦ Modality = 1
↦ Parameter term = 3xy5
↦ Determinacy <= 8
↦
poly f  = x^3+x*y^5+4*y^8-z^2;  // E[13]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = E[13]
↦ Normalform equation = x3+xy5+4y8
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = 4y8
↦ Determinacy <= 9
↦
poly f  = x^3+y^8+2*x*y^6+z^2;  // E[14]+
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = E[14]+
↦ Normalform equation = x3+2xy6+y8
↦ Milnor number = 14
↦ Modality = 1
↦ Parameter term = 2xy6
↦ Determinacy <= 9
↦
poly f  = x^3*y+y^5+5*x*y^4-z^2;  // Z[11]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = Z[11]
↦ Normalform equation = x3+5xy4+y5
↦ Milnor number = 11
↦ Modality = 1
↦ Parameter term = 5xy4
↦ Determinacy <= 5
↦
poly f  = x^3*y+x*y^4+6*x^2*y^3+z^2;  // Z[12]
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = Z[12]
↦ Normalform equation = x3y+6x2y3+xy4
↦ Milnor number = 12
↦ Modality = 1
↦ Parameter term = 6x2y3
↦ Determinacy <= 6
↦
poly f  = x^3*y-y^6+2*x*y^5-z^2;  // Z[13]-
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = Z[13]-
↦ Normalform equation = x3y+2xy5-y6
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = 2xy5
↦ Determinacy <= 6
↦
poly f  = x^4+y^5+x^2*y^3+z^2;  // W[12]+
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = W[12]+
↦ Normalform equation = x4+x2y3+y5
↦ Milnor number = 12
↦ Modality = 1
↦ Parameter term = x2y3
↦ Determinacy <= 5
↦
poly f  = -x^4+x*y^4+y^6-z^2;  // W[13]-
f=phi(f);
realclassify(f);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = W[13]-
↦ Normalform equation = -x4+xy4+y6
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = y6
↦ Determinacy <= 6
↦
poly p = x^4-x^2*y^2+5*y^4+x*y^3+x^3*y+z^2;  //X9++
f=phi(f);
realclassify(p);
↦ [1]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = X[9]++
↦ Normalform equation = x4+(a)*x2y2+y4
↦ Milnor number = 9
↦ Modality = 1
↦ Parameter term = (a)*x2y2
↦ Minimal polynomial = (18541a6-2454316a4+32984048a2-17909824)
↦ Interval = [10829268444323/1000000000000, 433170737773/40000000000]
↦ Determinacy <= 4
↦
↦ [2]:
↦    Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = X[9]++
↦ Normalform equation = x4+(a)*x2y2+y4
↦ Milnor number = 9
↦ Modality = 1
↦ Parameter term = (a)*x2y2
↦ Minimal polynomial = (18541a6-2454316a4+32984048a2-17909824)
↦ Interval = [-94106465721/125000000000, -752851725767/1000000000000]
↦ Determinacy <= 4
↦
```

See also: Section D.6.4.2 [classify], page 1646.

D.6.19.2 realmorsesplit

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Usage: realmorsesplit(f[, mu]); f poly, mu int

Return: a list consisting of the corank of f, the inertia index, an upper bound for the determinacy, and the residual form of f

Note: The characteristic of the basering must be zero, the monomial order must be local, f must be contained in maxideal(2) and the Milnor number of f must be finite.
The Milnor number of f can be provided as an optional parameter in order to avoid that it is computed again.

Example:

```
LIB "realclassify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
realmorsesplit(f);
↦ [1]:
↦    2
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    -x3+2/3xy2
```

See also: Section D.6.4.8 [morsesplit], page 1650.

D.6.19.3 milnornumber

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Usage: milnornumber(f); f poly

Return: Milnor number of f, or -1 if the Milnor number is not finite

Note: The monomial order must be local.

Example:

```
LIB "realclassify.lib";
ring r = 0, (x,y), ds;
poly f = x3+y4;
milnornumber(f);
↦ 6
```

D.6.19.4 determinacy

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Usage: determinacy(f[, mu]); f poly, mu int

Return: an upper bound for the determinacy of f

Note: The characteristic of the basering must be zero, the monomial order must be local, f must be contained in maxideal(1) and the Milnor number of f must be finite.
The Milnor number of f can be provided as an optional parameter in order to avoid that it is computed again.

Example:

```
LIB "realclassify.lib";
ring r = 0, (x,y), ds;
poly f = x3+xy3;
determinacy(f);
↦ 5
```

See also: Section 5.1.55 [highcorner], page 194; Section D.6.19.3 [milnornumber], page 1754.

D.6.19.5 addnondegeneratevariables

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Usage: addnondegeneratevariables(NFR); NFR NormalFormEquation

Return: Adds squares of the non-degenerate variables (i.e. var(cr+1), ..., var(nvars(basering)) for corank cr) to the normalform nf, with signs according to the inertia index.

Example:

```
LIB "realclassify.lib";
ring r = 0, (x,y,z), ds;
poly f = (x2+3y-2z)^2+xyz-(x-y3+x2z3)^3;
list NFR = realclassify(f);
NFR[1];
↦ Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = D[4]-
↦ Normalform equation = x2y-y3
↦ Milnor number = 4
↦ Modality = 0
↦ Determinacy <= 3
↦
addnondegeneratevariables(NFR[1]);
↦ Corank = 2
↦ Inertia index = 0
↦ Normalform equation of type = D[4]-
↦ Normalform equation = z2+x2y-y3
↦ Milnor number = 4
↦ Modality = 0
↦ Determinacy <= 3
↦
```

See also: Section D.6.19.2 [realmorsesplit], page 1753.

D.6.19.6 HilbertClassPoly

Procedure from library `realclassify.lib` (see Section D.6.19 [realclassify_lib], page 1745).

Return: the monic polynomial of degree h(D) in Z[X] of which jOft((D+sqr(D))/2) is a root

Assume: D is a negative discriminant

Note: k is input for the procedure "jOft",
5*k is input for the procedure "sqr",
10*k describes the number of decimals being calculated in the complex numbers

Example:

```
LIB "realclassify.lib";
ring r = 0,x,dp;
bigint D=-23;
HilbertClassPoly(D,50);
↦ x3+3491750x2-5151296875x+12771880859375
```

D.6.20 sing_lib

Library: sing.lib

Purpose: Invariants of Singularities

Authors: Gert-Martin Greuel, email: greuel@mathematik.uni-kl.de
Bernd Martin, email: martin@math.tu-cottbus.de

Procedures:

D.6.20.1 codim

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: codim(id1,id2); id1,id2 ideal or module, both must be standard bases

Return: int, which is:
1. the vectorspace dimension of id1/id2 if id2 is contained in id1 and if this number is finite
2. -1 if the dimension of id1/id2 is infinite
3. -2 if id2 is not contained in id1

Compute: consider the Hilbert series iv1(t) of id1 and iv2(t) of id2. If codim(id1,id2) is finite, q(t)=(iv2(t)-iv1(t))/(1-t)^n is rational, and the codimension is the sum of the coefficients of q(t) (n = dimension of basering).

Example:

```
LIB "sing.lib";
ring r  = 0,(x,y),dp;
ideal j = y6,x4;
ideal m = x,y;
attrib(m,"isSB",1);  //let Singular know that ideals are a standard basis
attrib(j,"isSB",1);
codim(m,j);          // should be 23 (Milnor number -1 of y7-x5)
↦ 23
```

D.6.20.2 deform

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: deform(id); id=ideal or poly

Return: matrix, columns are kbase of infinitesimal deformations

Example:

```
LIB "sing.lib";
ring r   = 32003,(x,y,z),ds;
ideal i  = xy,xz,yz;
matrix T = deform(i);
print(T);
↦ x,0,0,
↦ 0,0,z,
↦ 0,y,0
print(deform(x3+y5+z2));
↦ xy3,y3,xy2,y2,xy,y,x,1
```

D.6.20.3 dim_slocus

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: dim_slocus(i); i ideal or poly

Return: dimension of singular locus of i

Example:

```
LIB "sing.lib";
ring r  = 32003,(x,y,z),ds;
ideal i = x5+y6+z6,x2+2y2+3z2;
dim_slocus(i);
↦ 0
```

D.6.20.4 is_active

Procedure from library sing.lib (see Section D.6.20 [sing_lib], page 1755).

Usage: is_active(f,id); f poly, id ideal or module

Return: 1 if f is an active element modulo id (i.e. dim(id)=dim(id+f*R^n)+1, if id is a submodule of R^n) resp. 0 if f is not active. The basering may be a quotient ring

Note: regular parameters are active but not vice versa (id may have embedded components). proc is_reg tests whether f is a regular parameter

Example:

```
LIB "sing.lib";
ring r   =32003,(x,y,z),ds;
ideal i  = yx3+y,yz3+y3z;
poly f   = x;
is_active(f,i);
↦ 1
qring q  = std(x4y5);
poly f   = x;
module m = [yx3+x,yx3+y3x];
is_active(f,m);
↦ 0
```

D.6.20.5 is_ci

Procedure from library sing.lib (see Section D.6.20 [sing_lib], page 1755).

Usage: is_ci(i); i ideal

Return: intvec = sequence of dimensions of ideals (j[1],...,j[k]), for k=1,...,size(j), where j is minimal base of i. i is a complete intersection if last number equals nvars-size(i)

Note: dim(0-ideal) = -1. You may first apply simplify(i,10); in order to delete zeroes and multiples from set of generators
printlevel >=0: display comments (default)

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;                // display comments
ring r     = 32003,(x,y,z),ds;
ideal i    = x4+y5+z6,xyz,yx2+xz2+zy7;
is_ci(i);
↦ // complete intersection of dim 0
↦ // dim-sequence:
↦ 2,1,0
i          = xy,yz;
is_ci(i);
```

```
↦ // no complete intersection
↦ // dim-sequence:
↦ 2,2
printlevel = p;
```

D.6.20.6 is_is

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: is_is(id); id ideal or poly

Return: intvec = sequence of dimensions of singular loci of ideals generated by id[1]..id[i], k = 1..size(id);
dim(0-ideal) = -1;
id defines an isolated singularity if last number is 0

Note: printlevel >=0: display comments (default)

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 32003,(x,y,z),ds;
ideal i    = x2y,x4+y5+z6,yx2+xz2+zy7;
is_is(i);
↦ // dim of singular locus = 0
↦ // isolated singularity if last number is 0 in dim-sequence:
↦ 2,1,0
poly f     = xy+yz;
is_is(f);
↦ // dim of singular locus = 1
↦ // isolated singularity if last number is 0 in dim-sequence:
↦ 1
printlevel = p;
```

D.6.20.7 is_reg

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: is_reg(f,id); f poly, id ideal or module

Return: 1 if multiplication with f is injective modulo id, 0 otherwise

Note: Let R be the basering and id a submodule of R^n. The procedure checks injectivity of multiplication with f on R^n/id. The basering may be a quotient ring.

Example:

```
LIB "sing.lib";
ring r  = 32003,(x,y),ds;
ideal i = x8,y8;
ideal j = (x+y)^4;
i       = intersect(i,j);
poly f  = xy;
is_reg(f,i);
↦ 0
```

D.6.20.8 is_regs

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: is_regs(i[,id]); i poly, id ideal or module (default: id=0)

Return: 1 if generators of i are a regular sequence modulo id, 0 otherwise

Note: Let R be the basering and id a submodule of R^n. The procedure checks injectivity of multiplication with i[k] on R^n/id+i[1..k-1]. The basering may be a quotient ring.
printlevel >=0: display comments (default)
printlevel >=1: display comments during computation

Example:

```
LIB "sing.lib";
int p       = printlevel;
printlevel = 1;
ring r1     = 32003,(x,y,z),ds;
ideal i     = x8,y8,(x+y)^4;
is_regs(i);
↦ // checking whether element 1 is regular mod 1 .. 0
↦ // checking whether element 2 is regular mod 1 .. 1
↦ // checking whether element 3 is regular mod 1 .. 2
↦ // elements 1..2 are regular, 3 is not regular mod 1..2
↦ 0
module m    = [x,0,y];
i           = x8,(x+z)^4;;
is_regs(i,m);
↦ // checking whether element 1 is regular mod 1 .. 0
↦ // checking whether element 2 is regular mod 1 .. 1
↦ // elements are a regular sequence of length 2
↦ 1
printlevel = p;
```

D.6.20.9 locstd

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: locstd (id); id = ideal

Return: a standard basis for a local degree ordering

Note: the procedure homogenizes id w.r.t. a new 1st variable `@t@`, computes a SB w.r.t. (dp(1),dp) and substitutes `@t@` by 1.
Hence the result is a SB with respect to an ordering which sorts first w.r.t. the order and then refines it with dp. This is a local degree ordering.
This is done in order to avoid cancellation of units and thus be able to use option(contentSB);

Example:

```
LIB "sing.lib";
ring R = 0,(x,y,z),ds;
ideal i  = xyz+z5,2x2+y3+z7,3z5+y5;
locstd(i);
↦ _[1]=y5+3z5
↦ _[2]=3x4y3z8-4x3y3z9+6x2y4z9+3y5z10
↦ _[3]=3x4z13-4x3z14+6x2yz14+3y2z15
↦ _[4]=3x4yz12-4x3yz13+6x2y2z13+3y3z14
↦ _[5]=2x2z9+x2y2z8+y3z9
↦ _[6]=2x2y4z5+y7z5-3x2yz9
↦ _[7]=6y2z10-3x2y3z8+4xy3z9-3y4z9
↦ _[8]=3x2y2z8+3y3z9+2xy4z8
↦ _[9]=18z14-4xy6z8+3y7z8-9x2yz12
↦ _[10]=xyz+z5
↦ _[11]=3xz6-y4z5
↦ _[12]=3y3z6+2xy4z5-3xyz9
↦ _[13]=y4z5-2xz9-xy2z8
↦ _[14]=3z10+2xyz9+xy3z8
↦ _[15]=2x2z5+y3z5-xyz8
↦ _[16]=y4z-2xz5+yz8
↦ _[17]=3z6+2xyz5-y2z8
↦ _[18]=2x2+y3+z7
```

D.6.20.10 milnor

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: milnor(i); i ideal or poly

Return: Milnor number of i, if i is ICIS (isolated complete intersection singularity) in generic form, resp. -1 if not

Note: use proc nf_icis to put generators in generic form
printlevel >=1: display comments

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 2;
ring r     = 32003,(x,y,z),ds;
ideal j    = x5+y6+z6,x2+2y2+3z2,xyz+yx;
milnor(j);
↦ //sequence of discriminant numbers: 100,149,70
↦ 21
poly f     = x7+y7+(x-y)^2*x2y2+z2;
milnor(f);
↦ 28
printlevel = p;
```

D.6.20.11 nf_icis

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: nf_icis(i); i ideal

Return: ideal = generic linear combination of generators of i if i is an ICIS (isolated complete intersection singularity), return i if not

Note: this proc is useful in connection with proc milnor
printlevel >=0: display comments (default)

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 32003,(x,y,z),ds;
ideal i    = x3+y4,z4+yx;
nf_icis(i);
↦ // complete intersection of dim 1
↦ // dim-sequence:
↦ // dim of singular locus = 0
↦ // isolated singularity if last number is 0 in dim-sequence:
↦ // dim of singular locus = 0
↦ // isolated singularity if last number is 0 in dim-sequence:
↦ // ICIS in generic form after 1 genericity loop(s)
↦ _[1]=2xy+x3+y4+2z4
↦ _[2]=xy+z4
ideal j    = x3+y4,xy,yz;
nf_icis(j);
↦ // no complete intersection
↦ // dim-sequence:
↦ // no complete intersection
↦ _[1]=x3+y4
↦ _[2]=xy
↦ _[3]=yz
printlevel = p;
```

D.6.20.12 slocus

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: slocus(i); i ideal

Return: ideal of singular locus of i. Quotient rings and rings with integer coefficients are currently not supported.

Example:

```
LIB "sing.lib";
ring r  = 0,(u,v,w,x,y,z),dp;
ideal i = wx,wy,wz,vx,vy,vz,ux,uy,uz,y3-x2;;
slocus(i);
↦ _[1]=x
↦ _[2]=w
↦ _[3]=v
↦ _[4]=u
↦ _[5]=y2
```

D.6.20.13 qhspectrum

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: qhspectrum(f,w); f=poly, w=intvec

Assume: f is a weighted homogeneous isolated singularity w.r.t. the weights given by w; w must consist of as many positive integers as there are variables of the basering

Compute: the spectral numbers of the w-homogeneous polynomial f, computed in a ring of characteristic 0

Return: intvec d,s1,...,su where:
d = w-degree(f) and si/d = i-th spectral-number(f)
No return value if basering has parameters or if f is no isolated singularity, displays a warning in this case.

Example:

```
LIB "sing.lib";
ring r;
poly f=x3+y5+z2;
intvec w=10,6,15;
qhspectrum(f,w);
↦ 30,1,7,11,13,17,19,23,29
// the spectrum numbers are:
// 1/30,7/30,11/30,13/30,17/30,19/30,23/30,29/30
```

D.6.20.14 Tjurina

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: Tjurina(id[,<any>]); id=ideal or poly

Assume: id=ICIS (isolated complete intersection singularity)

Return: standard basis of Tjurina-module of id,
of type module if id=ideal, resp. of type ideal if id=poly. If a second argument is present (of any type) return a list:
[1] = Tjurina number,
[2] = k-basis of miniversal deformation,
[3] = SB of Tjurina module,
[4] = Tjurina module

Display: Tjurina number if printlevel >= 0 (default)

Note: Tjurina number = -1 implies that id is not an ICIS

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 0,(x,y,z),ds;
poly f     = x5+y6+z7+xyz;        // singularity T[5,6,7]
list T     = Tjurina(f,"");
↦ // Tjurina number = 16
show(T[1]);                       // Tjurina number, should be 16
↦ // int, size 1
↦ 16
show(T[2]);                       // basis of miniversal deformation
↦ // ideal, 16 generator(s)
↦ z6,
↦ z5,
↦ z4,
↦ z3,
↦ z2,
↦ z,
↦ y5,
↦ y4,
↦ y3,
↦ y2,
↦ y,
↦ x4,
↦ x3,
↦ x2,
↦ x,
↦ 1
show(T[3]);                       // SB of Tjurina ideal
↦ // ideal, 6 generator(s)
↦ xy+7z6,
↦ xz+6y5,
↦ yz+5x4,
↦ 5x5-6y6,
↦ y6,
↦ z7
show(T[4]); "";                   // Tjurina ideal
↦ // ideal, 4 generator(s)
↦ yz+5x4,
↦ xz+6y5,
↦ xy+7z6,
↦ xyz+x5+y6+z7
↦
ideal j    = x2+y2+z2,x2+2y2+3z2;
show(kbase(Tjurina(j)));          // basis of miniversal deformation
↦ // Tjurina number = 5
↦ // module, 5 generator(s)
↦ [z]
↦ [y]
↦ [x]
↦ [1]
↦ [0,1]
hilb(Tjurina(j));                 // Hilbert series of Tjurina module
↦ // Tjurina number = 5
↦ (-3t4+7t3-3t2-3t+2) / (1-t)^3
↦ (3t+2) / (1-t)^0
↦ // dimension (local)   = 0
↦ // multiplicity = 5
printlevel = p;
```

D.6.20.15 tjurina

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: tjurina(id); id=ideal or poly

Assume: id=ICIS (isolated complete intersection singularity)

Return: int = Tjurina number of id

Note: Tjurina number = -1 implies that id is not an ICIS

Example:

```
LIB "sing.lib";
ring r=32003,(x,y,z),(c,ds);
```

```
ideal j=x2+y2+z2,x2+2y2+3z2;
tjurina(j);
↦ 5
```

D.6.20.16 T_1

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: T_1(id[,<any>]); id = ideal or poly

Return: T_1(id): of type module/ideal if id is of type ideal/poly. We call T_1(id) the T_1-module of id. It is a std basis of the presentation of 1st order deformations of P/id, if P is the basering. If a second argument is present (of any type) return a list of 3 modules:
[1]= T_1(id)
[2]= generators of normal bundle of id, lifted to P
[3]= module of relations of [2], lifted to P
(note: transpose[3]*[2]=0 mod id)
The list contains all non-easy objects which must be computed to get T_1(id).

Display: k-dimension of T_1(id) if printlevel >= 0 (default)

Note: T_1(id) itself is usually of minor importance. Nevertheless, from it all relevant information can be obtained. The most important are probably vdim(T_1(id)); (which computes the Tjurina number), hilb(T_1(id)); and kbase(T_1(id)).
If T_1 is called with two arguments, then matrix([2])*(kbase([1])) represents a basis of 1st order semiuniversal deformation of id (use proc 'deform', to get this in a direct way).
For a complete intersection the proc Tjurina is faster.

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 32003,(x,y,z),(c,ds);
ideal i    = xy,xz,yz;
module T   = T_1(i);
↦ // dim T_1 = 3
vdim(T);                      // Tjurina number = dim_K(T_1), should be 3
↦ 3
list L=T_1(i,"");
↦ // dim T_1 = 3
module kB  = kbase(L[1]);
print(matrix(L[2])*matrix(kB)); // basis of 1st order miniversal deformation
↦ 0,0,0,
↦ z,0,0,
↦ 0,y,z
show(L[2]);                   // presentation of normal bundle
↦ // module, 6 generator(s)
↦ [x]
↦ [y,z]
↦ [0,x,y]
↦ [0,z]
↦ [0,0,y]
↦ [0,0,z]
```

```
print(L[3]);                    // relations of i
↦ z, 0,
↦ -y,y,
↦ 0, -x
print(transpose(matrix(L[3]))*matrix(L[2]));  // should be 0 (mod i)
↦ xz,0, -xy,-yz,0,  0,
↦ 0, yz,0,  yz, -xy,-xz
printlevel = p;
```

D.6.20.17 T_2

Procedure from library sing.lib (see Section D.6.20 [sing_lib], page 1755).

Usage: T_2(id[,<any>]); id = ideal

Return: T_2(id): T_2-module of id . This is a std basis of a presentation of the module of obstructions of R=P/id, if P is the basering. If a second argument is present (of any type) return a list of 4 modules and 1 ideal:
[1]= T_2(id)
[2]= standard basis of id (ideal)
[3]= module of relations of id (=1st syzygy module of id)
[4]= presentation of syz/kos
[5]= relations of Hom_P([3]/kos,R), lifted to P
The list contains all non-easy objects which must be computed to get T_2(id).

Display: k-dimension of T_2(id) if printlevel >= 0 (default)

Note: The most important information is probably vdim(T_2(id)). Use proc miniversal to get equations of the miniversal deformation.

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring  r    = 32003,(x,y),(c,dp);
ideal j    = x6-y4,x6y6,x2y4-x5y2;
module T   = T_2(j);
↦ // dim T_2 = 6
vdim(T);
↦ 6
hilb(T);"";
↦ (t5-t3-t2+1) / (1-t)^2
↦ (t3+2t2+2t+1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 6
↦
ring r1    = 0,(x,y,z),dp;
ideal id   = xy,xz,yz;
list L     = T_2(id,"");
↦ // dim T_2 = 0
vdim(L[1]);                             // vdim of T_2
↦ 0
print(L[3]);                            // syzygy module of id
↦ -z,-z,
↦ y, 0,
↦ 0, x
printlevel = p;
```

D.6.20.18 T_12

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: T_12(i[,any]); i = ideal

Return: T_12(i): list of 2 modules:
* standard basis of T_1-module =T_1(i), 1st order deformations
* standard basis of T_2-module =T_2(i), obstructions of R=P/i
If a second argument is present (of any type) return a list of 9 modules, matrices, integers:
[1]= standard basis of T_1-module
[2]= standard basis of T_2-module
[3]= vdim of T_1
[4]= vdim of T_2
[5]= matrix, whose cols present infinitesimal deformations
[6]= matrix, whose cols are generators of relations of i(=syz(i))
[7]= matrix, presenting Hom_P(syz/kos,R), lifted to P
[8]= presentation of T_1-module, no std basis
[9]= presentation of T_2-module, no std basis

Display: k-dimension of T_1 and T_2 if printlevel >= 0 (default)

Note: Use proc miniversal from deform.lib to get miniversal deformation of i, the list contains all objects used by proc miniversal.

Example:

```
LIB "sing.lib";
int p      = printlevel;
printlevel = 1;
ring r     = 199,(x,y,z,u,v),(c,ws(4,3,2,3,4));
ideal i    = xz-y2,yz2-xu,xv-yzu,yu-z3,z2u-yv,zv-u2;
//a cyclic quotient singularity
list L     = T_12(i,1);
↦ // dim T_1 = 5
↦ // dim T_2 = 3
print(L[5]);             //matrix of infin. deformations
↦ 0,  0,  0,  0,  0,
↦ yz, y,  z2, 0,  0,
↦ -z3,-z2,-zu,yz, yu,
↦ -z2,-z, -u, 0,  0,
↦ zu, u,  v,  -z2,-zu,
↦ 0,  0,  0,  u,  v
printlevel = p;
```

D.6.20.19 tangentcone

Procedure from library `sing.lib` (see Section D.6.20 [sing_lib], page 1755).

Usage: tangentcone(id [,n]); id = ideal, n = int

Return: the tangent cone of id

Note: The procedure works for any monomial ordering.
If n=0 use std w.r.t. local ordering ds, if n=1 use locstd.

Example:

```
LIB "sing.lib";
ring R = 0,(x,y,z),ds;
ideal i  = 7xyz+z5,x2+y3+z7,5z5+y5;
tangentcone(i);
↦ _[1]=x2
↦ _[2]=7xyz
↦ _[3]=y5+5z5
↦ _[4]=7y4z
↦ _[5]=35z6
```

D.6.21 spcurve_lib

Library: spcurve.lib

Purpose: Deformations and Invariants of CM-codim 2 Singularities

Author: Anne Fruehbis-Krueger, anne@mathematik.uni-kl.de

Procedures:

D.6.21.1 isCMcod2

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: isCMcod2(i); i an ideal

Return: presentation matrix of i, if i is Cohen-Macaulay of codimension 2
a zero matrix otherwise

Example:

```
LIB "spcurve.lib";
ring r=32003,(x,y,z),ds;
ideal i=xz,yz,x^3-y^4;
print(isCMcod2(i));
↦ -y,-x2,
↦ x, y3,
↦ 0, z
```

D.6.21.2 CMtype

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: CMtype(i); i an ideal, CM of codimension 2

Return: Cohen-Macaulay type of i (integer)
(-1, if i is not Cohen-Macaulay of codimension 2)

Example:

```
LIB "spcurve.lib";
ring r=32003,(x,y,z),ds;
ideal i=xy,xz,yz;
CMtype(i);
↦ 2
```

D.6.21.3 matrixT1

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: matrixT1(M,n); M matrix, n integer

Assume: M is a presentation matrix of an ideal i, CM of codimension 2; consider i as a family of ideals in a ring in the first n variables where the remaining variables are considered as parameters

Return: list consisting of the k x (k+1) matrix M and a module K_M such that T1=Mat(k,k+1;R)/K_M is the space of first order deformations of i

Example:

```
LIB "spcurve.lib";
ring r=32003,(x(1),x(2),x(3)),ds;
ideal curve=x(1)*x(2),x(1)*x(3),x(2)*x(3);
matrix M=isCMcod2(curve);
matrixT1(M,3);
↦ [1]:
↦    _[1,1]=0
↦    _[1,2]=-x(3)
↦    _[2,1]=-x(2)
↦    _[2,2]=x(2)
↦    _[3,1]=x(1)
↦    _[3,2]=0
↦ [2]:
↦    _[1]=gen(5)
↦    _[2]=gen(4)-gen(3)
↦    _[3]=-gen(2)
↦    _[4]=x(1)*gen(5)-x(2)*gen(3)
↦    _[5]=x(1)*gen(6)-x(2)*gen(4)
↦    _[6]=x(2)*gen(3)-x(3)*gen(1)
↦    _[7]=x(2)*gen(4)-x(3)*gen(2)
↦    _[8]=-x(3)*gen(2)
↦    _[9]=x(2)*gen(2)-x(2)*gen(1)
↦    _[10]=x(1)*gen(1)
↦    _[11]=-x(3)*gen(4)
↦    _[12]=x(2)*gen(4)-x(2)*gen(3)
↦    _[13]=x(1)*gen(3)
↦    _[14]=-x(3)*gen(6)
↦    _[15]=x(2)*gen(6)-x(2)*gen(5)
↦    _[16]=x(1)*gen(5)
```

D.6.21.4 semiCMcod2

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: semiCMcod2(M,t1[,s]); M matrix, t1 module, s any

Assume: M is a presentation matrix of an ideal i, CM of codimension 2, and t1 is a presentation of the space of first order deformations of i ((M,t1) as returned by the procedure matrixT1)

Return: new ring in which the ideal semi describing the semiuniversal deformation of i;
if the optional third argument is given, the perturbation matrix of the semiuniversal deformation is returned instead of the ideal.

Note: The current basering should not contain any variables named A(j) where j is some integer!

Example:

```
LIB "spcurve.lib";
ring r=32003,(x(1),x(2),x(3)),ds;
ideal curve=x(1)*x(2),x(1)*x(3),x(2)*x(3);
matrix M=isCMcod2(curve);
list l=matrixT1(M,3);
def rneu=semiCMcod2(l[1],std(l[2]));
setring rneu;
semi;
↦ semi[1]=A(2)*A(3)-x(2)*A(3)-x(1)*x(2)
↦ semi[2]=A(1)*A(3)+x(1)*x(3)
↦ semi[3]=-x(2)*A(1)-x(3)*A(2)+x(2)*x(3)
```

D.6.21.5 discr

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: discr(sem,n); sem ideal, n integer

Assume: sem is the versal deformation of an ideal of codimension 2.
The first n variables of the ring are treated as variables all the others as parameters.

Return: ideal describing the discriminant

Note: This is not a powerful algorithm!

Example:

```
LIB "spcurve.lib";
ring r=32003,(x(1),x(2),x(3)),ds;
ideal curve=x(1)*x(2),x(1)*x(3),x(2)*x(3);
matrix M=isCMcod2(curve);
list l=matrixT1(M,3);
def rneu=semiCMcod2(l[1],std(l[2]));
setring rneu;
discr(semi,3);
↦ _[1]=A(1)*A(2)*A(3)
```

D.6.21.6 qhmatrix

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: qhmatrix(M); M a k x (k+1) matrix

Return: list, consisting of an integer vector containing the weights of the variables of the basering and an integer matrix giving the weights of the entries of M, if M is quasihomogeneous; zero integer vector and zero integer matrix, if M is not quasihomogeneous, i.e. does not allow row and column weights

Example:

```
LIB "spcurve.lib";
ring r=0,(x,y,z),ds;
matrix M[3][2]=z,0,y,x,x^3,y;
qhmatrix(M);
```

```
↦ [1]:
↦    1,2,1
↦ [2]:
↦    1,0,
↦    2,1,
↦    3,2
pmat(M);
↦ z,  0,
↦ y,  x,
↦ x3, y
```

D.6.21.7 relweight

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: relweight(N,W,a); N matrix, W intmat, a intvec

Assume: N is a non-zero matrix
W is an integer matrix of the same size as N
a is an integer vector giving the weights of the variables

Return: integer, max(a-weighted order(N_ij) - W_ij | all entries ij)
string "ERROR" if sizes do not match

Example:

```
LIB "spcurve.lib";
ring r=32003,(x,y,z),ds;
matrix N[2][3]=z,0,y,x,x^3,y;
intmat W[2][3]=1,1,1,1,1,1;
intvec a=1,1,1;
relweight(N,W,a);
↦ 2
```

D.6.21.8 posweight

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: posweight(M,t1,n[,s]); M matrix, t1 module, n int, s string
n=0 : all deformations of non-negative weight
n=1 : only non-constant deformations of non-negative weight
n=2 : all deformations of positive weight

Assume: M is a presentation matrix of a Cohen-Macaulay codimension 2 ideal and t1 is its T1 space in matrix notation

Return: new ring containing a list posw, consisting of a presentation matrix describing the deformation given by the generators of T1 of non-negative/positive weight and the weight vector for the new variables

Note: The current basering should not contain any variables named T(i) where i is some integer!

Example:

```
LIB "spcurve.lib";
ring r=32003,(x(1),x(2),x(3)),ds;
```

```
ideal curve=(x(3)-x(1)^2)*x(3),(x(3)-x(1)^2)*x(2),x(2)^2-x(1)^7*x(3);
matrix M=isCMcod2(curve);
list l=matrixT1(M,3);
def rneu=posweight(l[1],std(l[2]),0);
setring rneu;
pmat(posw[1]);
↦ T(2)+x(1)*T(1), -x(3)+x(1)^2,
↦ -x(3),          x(2),
↦ x(2),           -x(1)^7
posw[2];
↦ 3,1
```

D.6.21.9 KSpencerKernel

Procedure from library `spcurve.lib` (see Section D.6.21 [spcurve_lib], page 1767).

Usage: KSpencerKernel(M[,s][,v]); M matrix, s string, v intvec
optional parameters (please specify in this order, if both are present):
* s = first of the names of the new rings
e.g. "R" leads to ring names R and R1
* v of size n(n+1) leads to the following module ordering
gen(v[1]) > gen(v[2]) > ... > gen(v[n(n+1)]) where the matrix entry ij corresponds to gen((i-1)*n+j)

Assume: M is a quasihomogeneous n x (n+1) matrix where the n minors define an isolated space curve singularity

Return: new ring containing the coefficient matrix KS representing the kernel of the Kodaira-Spencer map of the family of non-negative deformations having the given singularity as special fibre

Note: * the initial basering should not contain variables with name e(i) or T(i), since those variable names will internally be used by the script
* setting an intvec with 5 entries and name watchProgress shows the progress of the computations:
watchProgress[1]>0 => option(prot) in groebner commands
watchProgress[2]>0 => trace output for highcorner
watchProgress[3]>0 => output of deformed matrix
watchProgress[4]>0 => result of elimination step
watchProgress[4]>1 => trace output of multiplications with xyz and subsequent reductions
watchProgress[5]>0 => matrix representing the kernel using print

Example:

```
LIB "spcurve.lib";
ring r=0,(x,y,z),ds;
matrix M[3][2]=z-x^7,0,y^2,z,x^9,y;
def rneu=KSpencerKernel(M,"ar");
setring rneu;
basering;
↦ // coefficients: QQ
↦ // number of vars : 17
↦ //        block   1 : ordering Ws
↦ //                  : names    e(1) e(2) e(3) e(4) e(5) e(6) x y z
↦ //                  : weights  -21 -10 -32 -21 -27 -16 3 16 21
↦ //        block   2 : ordering wp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) T(6) T(7) T(8)
↦ //                  : weights     8    5    2   10    7    4    1    2
↦ //        block   3 : ordering C
print(KS);
↦ T(7),   0,      0,      0,     0,     0,     0,     0,
↦ KS[2,1],6*T(3), 3*T(7), 0,     0,     0,     0,     0,
↦ KS[3,1],KS[3,2],KS[3,3],6*T(3),3*T(7),0,     0,     0,
↦ 10*T(4),8*T(1), 7*T(5), 5*T(2),4*T(6),2*T(8),2*T(3),T(7)
```

D.6.22 spectrum_lib

Library: spectrum.lib

Purpose: Singularity Spectrum for Nondegenerate Singularities

Author: S. Endrass

Procedures:

D.6.22.1 spectrumnd

Procedure from library `spectrum.lib` (see Section D.6.22 [spectrum_lib], page 1772).

Usage: spectrumnd(f[,1]); poly f

Assume: basering has characteristic 0 and local ordering,
f has isolated singularity at 0 and nondegenerate principal part

Return:

list S:
  ideal S[1]: spectral numbers in increasing order
  intvec S[2]:
    int S[2][i]: multiplicity of spectral number S[1][i]

Note: if a second argument 1 is given,
no test for a degenerate principal part will be done
SEE_ALSO: gmssing_lib

Example:

```
LIB "spectrum.lib";
ring R=0,(x,y),ds;
poly f=x^31+x^6*y^7+x^2*y^12+x^13*y^2+y^29;
list s=spectrumnd(f);
size(s[1]);
↦ 174
s[1][22];
↦ -27/58
s[2][22];
↦ 2
```

D.6.23 surfacesignature_lib

Library: surfacesignature.lib

Purpose: signature of surface singularity

Authors: Gerhard Pfister pfister@mathematik.uni-kl.de
Muhammad Ahsan Banyamin ahsanbanyamin@gmail.com
Stefan Steidel steidel@mathematik.uni-kl.de

Overview: A library for computing the signature of irreducible surface singularity. The signature of a surface singularity is defined in [3]. The algorithm we use has been proposed in [9].
Let g in C[x,y] define an isolated curve singularity at 0 in C^2 and f:=z^N+g(x,y). The zero-set V:=V(f) in C^3 of f has an isolated singularity at 0. For a small e>0 let V_e:=V(f-e) in C^3 be the Milnor fibre of (V,0) and s: H_2(V_e,R) x H_2(V_e,R) —> R be the intersection form (cf. [1],[7]). H_2(V_e,R) is an m-dimensional R-vector space, m the Milnor number of (V,0) (cf. [1],[4],[5],[6]), and s is a symmetric bilinear form. Let sigma(f) be the signature of s, called the signature of the surface singularity (V,0). Formulaes to compute the signature are given by Nemethi (cf. [8],[9]) and van Doorn, Steenbrink (cf. [2]).
We have implemented three approaches using Puiseux expansions, the resolution of singularities resp. the spectral pairs of the singularity.

References:

[1] Arnold, V.I.; Gusein-Zade, S.M.; Varchenko, A.N.: Singularities of Differentiable Mappings. Vol. 1,2, Birkh"auser (1988). [2] van Doorn, M.G.M.; Steenbrink, J.H.M.: A supplement to the monodromy theorem. Abh. Math. Sem. Univ. Hamburg 59, 225-233 (1989). [3] Durfee, A.H.: The Signature of Smoothings of Complex Surface Singularities. Mathematische Annalen 232, 85-98 (1978). [4] de Jong, T.; Pfister, G.: Local Analytic Geometry. Vieweg (2000). [5] Kerner, D.; Nemethi, A.: The Milnor fibre signature is not semi-continous. arXiv:0907.5252 (2009).
[6] Kulikov, V.S.: Mixed Hodge Structures and Singularities. Cambridge Tracts in Mathematics 132, Cambridge University Press (1998). [7] Nemethi, A.: The real Seifert form and the spectral pairs of isolated hypersurface singularities. Compositio Mathematica 98, 23-41 (1995). [8] Nemethi, A.: Dedekind sums and the signature of f(x,y)+z^N. Selecta Mathematica, New series, Vol. 4, 361-376 (1998).
[9] Nemethi, A.: The Signature of f(x,y)+z^$. Proceedings of Real and Complex Singularities (C.T.C. Wall's 60th birthday meeting, Liverpool (England), August 1996), London Math. Soc. Lecture Notes Series 263, 131–149 (1999).

Procedures:

D.6.23.1 signatureBrieskorn

Procedure from library `surfacesignature.lib` (see Section D.6.23 [surfacesignature_lib], page 1772).

Usage: signatureBrieskorn(a1,a2,a3); a1,a2,a3 = integers

Return: signature of Brieskorn singularity x^a1+y^a2+z^a3

Example:

```
LIB "surfacesignature.lib";
ring R = 0,x,dp;
signatureBrieskorn(11,3,5);
↦ -48
```

D.6.23.2 signaturePuiseux

Procedure from library `surfacesignature.lib` (see Section D.6.23 [surfacesignature_lib], page 1772).

Usage: signaturePuiseux(N,f); N = int, f = irreducible poly in 2 variables

Return: signature of surface singularity defined by z^N + f(x,y) = 0

Example:

```
LIB "surfacesignature.lib";
ring r = 0,(x,y),dp;
int N  = 3;
poly f = x15-21x14+8x13y-6x13-16x12y+20x11y2-x12+8x11y-36x10y2
+24x9y3+4x9y2-16x8y3+26x7y4-6x6y4+8x5y5+4x3y6-y8;
signaturePuiseux(N,f);
↦ -92
```

D.6.23.3 signatureNemethi

Procedure from library `surfacesignature.lib` (see Section D.6.23 [surfacesignature_lib], page 1772).

Usage: signatureNemethi(N,f); N = integer, f = reduced poly in 2 variables, # empty or 1,2,3
- if #[1] = 1 then resolution of singularity is used
- if #[1] = 2 then spectral pairs are used
- if # is empty then both upper variants are used in parallel and the fastest returns the result

Return: signature of surface singularity defined by z^N + f(x,y) = 0

Remark: computes the signature of some special surface singularities

Example:

```
LIB "surfacesignature.lib";
ring r = 0,(x,y),dp;
int N  = 3;
poly f = x15-21x14+8x13y-6x13-16x12y+20x11y2-x12+8x11y-36x10y2
+24x9y3+4x9y2-16x8y3+26x7y4-6x6y4+8x5y5+4x3y6-y8;
signatureNemethi(N,f,1);
↦ -92
printlevel = 1;
signatureNemethi(N,f);
↦ Resolution of singularity has been used.
↦ -92
```

D.7 Invariant theory

D.7.1 finvar_lib

Library: finvar.lib

Purpose: Invariant Rings of Finite Groups

Author: Agnes E. Heydtmann, contact via Wolfram Decker: decker@mathematik.uni-kl.de Simon A. King, email: simon.king@nuigalway.ie

Overview: A library for computing polynomial invariants of finite matrix groups and generators of related varieties. The algorithms are based on B. Sturmfels, G. Kemper, S. King and W. Decker et al..

Procedures:

D.7.1.1 invariant_ring

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_ring(G1,G2,...[,flags]);
G1,G2,...: <matrices> generating a finite matrix group, flags: an optional <intvec> with three entries: if the first one equals 0, the program attempts to compute the Molien series and Reynolds operator, if it equals 1, the program is told that the Molien series should not be computed, if it equals -1 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003) and if the first one is anything else, it means that the characteristic of the base field divides the group order (i.e. it will not even be attempted to compute the Reynolds operator or Molien series), the second component should give the size of intervals between canceling common factors in the expansion of Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field occurs not among the coefficients)

Return: primary and secondary invariants for any matrix representation of a finite group

Display: information about the various stages of the program if the third flag does not equal 0

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)). In the
non-modular case secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo the primary invariants, mapping to invariants with the Reynolds operator and using those or their power products such that they are linearly independent modulo the primary invariants (see "Some Algorithms in Invariant Theory of Finite Groups" by Kemper and Steel (1997)). In the modular case they are generated according to "Generating Invariant Rings of Finite Groups over Arbitrary Fields" by Kemper (1996).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix P,S,IS=invariant_ring(A);
print(P);
↦ z2,x2+y2,x2y2
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
print(IS);
↦ xyz,x2z-y2z,x3y-xy3
```

D.7.1.2 invariant_ring_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_ring_random(G1,G2,...,r[,flags]);
G1,G2,...: <matrices> generating a finite matrix group, r: an <int> where -|r| to |r| is the range of coefficients of random combinations of bases elements that serve as primary invariants, flags: an optional <intvec> with three entries: if the first equals 0, the program attempts to compute the Molien series and Reynolds operator, if it equals 1, the program is told that the Molien series should not be computed, if it equals -1 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003) and if the first one is anything else, then the characteristic of the base field divides the group order (i.e. we will not even attempt to compute the Reynolds operator or Molien series), the second component should give the size of intervals between canceling common factors in the expansion of the Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field does not occur among the coefficients)

Return: primary and secondary invariants for any matrix representation of a finite group

Display: information about the various stages of the program if the third flag does not equal 0

Theory: is the same as for invariant_ring except that random combinations of basis elements are chosen as candidates for primary invariants and hopefully they lower the dimension of the previously found primary invariants by the right amount.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix P,S,IS=invariant_ring_random(A,1);
print(P);
↦ z2,x2+y2,x4+y4-z4
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
print(IS);
↦ xyz,x2z-y2z,x3y-xy3
```

D.7.1.3 primary_invariants

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_invariants(G1,G2,...[,flags]);
G1,G2,...: <matrices> generating a finite matrix group, flags: an optional <intvec> with three entries, if the first one equals 0 (also the default), the programme attempts to compute the Molien series and Reynolds operator, if it equals 1, the programme is told that the Molien series should not be computed, if it equals -1 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003) and if the first one is anything else, it means that the characteristic of the base field divides the group order, the second component should give the size of intervals between canceling common factors in the expansion of the Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given

to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field occurs not among the coefficients)

Display: information about the various stages of the programme if the third flag does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and if computable Reynolds operator (type <matrix>) and Molien series (type <matrix>) or ring name (type string) where the Molien series can be found in the char p case; if the first flag is 1 and we are in the non-modular case then an <intvec> is returned giving some of the degrees where no non-trivial homogeneous invariants can be found

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A);
print(L[1]);
↦ z2,x2+y2,x2y2
```

D.7.1.4 primary_invariants_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_invariants_random(G1,G2,...,r[,flags]);
G1,G2,...: <matrices> generating a finite matrix group, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, flags: an optional <intvec> with three entries, if the first one equals 0 (also the default), the programme attempts to compute the Molien series and Reynolds operator, if it equals 1, the programme is told that the Molien series should not be computed, if it equals -1 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003) and if the first one is anything else, it means that the characteristic of the base field divides the group order, the second component should give the size of intervals between canceling common factors in the expansion of the Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field does not occur among the coefficients)

Display: information about the various stages of the programme if the third flag does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and if computable Reynolds operator (type <matrix>) and Molien series (type <matrix>), if the first flag is 1 and we are in the non-modular case then an <intvec> is returned giving some of the degrees where no non-trivial homogeneous invariants can be found

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants_random(A,1);
print(L[1]);
↦ z2,x2+y2,x4+y4-z4
```

D.7.1.5 invariant_algebra_reynolds

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_algebra_reynolds(REY[,v]);
REY: a gxn <matrix> representing the Reynolds operator of a finite matrix group, where g ist the group order and n is the number of variables of the basering;
v: an optional <int>

Return: A minimal homogeneous generating set of the invariant ring, type <matrix>

Assume: We are in the non-modular case, i.e., the characteristic of the basering does not divide the group order;
REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants()

Display: Information on the progress of computations if v does not equal 0

Theory: We do an incremental search in increasing degree d. Generators of the invariant ring are found among the Reynolds images of monomials of degree d. The generators are chosen by Groebner basis techniques (see S. King: Minimal generating sets of non-modular invariant rings of finite groups).

Note: invariant_algebra_reynolds should not be used in rings with weighted orders.

Example:

```
LIB "finvar.lib";
ring R=0,(a,b,c,d),dp;
matrix A[4][4]=
0,0,1,0,
0,0,0,1,
1,0,0,0,
0,1,0,0;
list L = group_reynolds(A);
matrix G = invariant_algebra_reynolds(L[1],1);
↦   We have  4  relevant monomials in degree  1
↦     We found generator number  1  in degree  1
↦     We found generator number  2  in degree  1
↦ Computing Groebner basis up to the new degree 2
↦   We have  3  relevant monomials in degree  2
↦     We found generator number  3  in degree  2
↦     We found generator number  4  in degree  2
↦     We found generator number  5  in degree  2
↦ Computing Groebner basis up to the new degree 3
↦ We found the degree bound 2
↦ We went beyond the degree bound, so, we are done!
G;
↦ G[1,1]=b+d
↦ G[1,2]=a+c
↦ G[1,3]=b2+d2
↦ G[1,4]=ab+cd
↦ G[1,5]=a2+c2
```

See also: Section D.7.1.6 [invariant_algebra_perm], page 1779.

D.7.1.6 invariant_algebra_perm

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_algebra_perm(GEN[,v]);
GEN: a list of generators of a permutation group. It is given in disjoint cycle form, where trivial cycles can be omitted; e.g., the generator (1,2)(3,4)(5) is given by <list(list(1,2),list(3,4))>.
v: an optional <int>

Return: A minimal homogeneous generating set of the invariant ring of the group presented by GEN, type <matrix>

Assume: We are in the non-modular case, i.e., the characteristic of the basering does not divide the group order. Note that the function does not verify whether this assumption holds or not

Display: Information on the progress of computations if v does not equal 0

Theory: We do an incremental search in increasing degree d. Generators of the invariant ring are found among the orbit sums of degree d. The generators are chosen by Groebner basis techniques (see S. King: Minimal generating sets of non-modular invariant rings of finite groups).

Note: invariant_algebra_perm should not be used in rings with weighted orders.

Example:

```
LIB "finvar.lib";
ring R=0,(a,b,c,d),dp;
def GEN=list(list(list(1,3),list(2,4)));
matrix G = invariant_algebra_perm(GEN,1);
↦ Searching generators in degree 1
↦   We have  2  orbit sums of degree  1
↦     We found generator number  1  in degree  1
↦     We found generator number  2  in degree  1
↦ Computing Groebner basis up to the new degree 2
↦ Searching generators in degree 2
↦   We have  3  orbit sums of degree  2
↦     We found generator number  3  in degree  2
↦     We found generator number  4  in degree  2
↦     We found generator number  5  in degree  2
↦ Computing Groebner basis up to the new degree 3
↦ We found the degree bound 2
↦ We went beyond the degree bound, so, we are done!
G;
↦ G[1,1]=b+d
↦ G[1,2]=a+c
↦ G[1,3]=b2+d2
↦ G[1,4]=ab+cd
↦ G[1,5]=a2+c2
```

See also: Section D.7.1.5 [invariant_algebra_reynolds], page 1778.

D.7.1.7 cyclotomic

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: cyclotomic(i); i integer > 0

Returns: the i-th cyclotomic polynomial (type <poly>) as one in the first ring variable

Theory: x^i-1 is divided by the j-th cyclotomic polynomial where j takes on the value of proper divisors of i

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
print(cyclotomic(25));
↦ x20+x15+x10+x5+1
```

D.7.1.8 group_reynolds

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: group_reynolds(G1,G2,...[,v]);
G1,G2,...: nxn <matrices> generating a finite matrix group, v: an optional <int>

Assume: n is the number of variables of the basering, g the number of group elements

Return: a <list>, the first list element will be a gxn <matrix> representing the Reynolds operator if we are in the non-modular case; if the characteristic is >0, minpoly==0 and the finite group non-cyclic the second list element is an <int> giving the lowest common multiple of the matrix group elements' order (used in molien); in general all other list elements are nxn <matrices> listing all elements of the finite group

Display: information if v does not equal 0

Theory: The entire matrix group is generated by getting all left products of generators with the new elements from the last run through the loop (or the generators themselves during the first run). All the ones that have been generated before are thrown out and the program terminates when no new elements found in one run. Additionally each time a new group element is found the corresponding ring mapping of which the Reynolds operator is made up is generated. They are stored in the rows of the first return value.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
print(L[1]);
↦ y, -x,-z,
↦ -x,-y,z,
↦ -y,x, -z,
↦ x, y, z
print(L[2..size(L)]);
↦ 0, 1,0,
↦ -1,0,0,
↦ 0, 0,-1
↦  -1,0, 0,
↦ 0, -1,0,
↦ 0, 0, 1
↦  0,-1,0,
↦ 1,0, 0,
↦ 0,0, -1
↦  1,0,0,
↦ 0,1,0,
↦ 0,0,1
```

D.7.1.9 molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: molien(G1,G2,...[,ringname,lcm,flags]);
G1,G2,...: nxn <matrices>, all elements of a finite matrix group, Finvar::newring: a new ring of characteristic 0 for the Molien series in case of prime characteristic, lcm: an <int>
giving the lowest common multiple of the elements' orders in case of prime characteristic, minpoly==0 and a non-cyclic group, flags: an optional <intvec> with three components: if the first element is not equal to 0 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003), the second component should give the size of intervals between canceling common factors in the expansion of the Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field does not occur among the coefficients)

Assume: n is the number of variables of the basering, G1,G2... are the group elements generated by group_reynolds(), lcm is the second return value of group_reynolds()

Return: in case of characteristic 0 a 1x2 <matrix> giving enumerator and denominator of Molien series; in case of prime characteristic a ring with the name newring of characteristic 0 is created where the same Molien series (named M) is stored

Display: information if the third component of flags does not equal 0

Theory: In characteristic 0 the terms 1/det(1-xE) for all group elements of the Molien series are computed in a straight forward way. In prime characteristic a Brauer lift is involved. The returned matrix gives enumerator and denominator of the expanded version where common factors have been canceled.

Example:

```
LIB "finvar.lib";
"         note the case of prime characteristic";
↦           note the case of prime characteristic
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
matrix M=molien(L[2..size(L)]);
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
```

```
ring S=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
molien(L[2..size(L)],"");
setring Finvar::newring;
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
setring S;
kill Finvar::newring;
```

D.7.1.10 reynolds_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: reynolds_molien(G1,G2,...[,ringname,flags]);
G1,G2,...: nxn <matrices> generating a finite matrix group, Finvar::newring: a new ring of characteristic 0 for the Molien series in case of prime characteristic, flags: an optional <intvec> with three components: if the first element is not equal to 0 characteristic 0 is simulated, i.e. the Molien series is computed as if the base field were characteristic 0 (the user must choose a field of large prime characteristic, e.g. 32003) the second component should give the size of intervals between canceling common factors in the expansion of the Molien series, 0 (the default) means only once after generating all terms, in prime characteristic also a negative number can be given to indicate that common factors should always be canceled when the expansion is simple (the root of the extension field does not occur among the coefficients)

Assume: n is the number of variables of the basering, G1,G2... are the group elements generated by group_reynolds(), g is the size of the group

Return: a gxn <matrix> representing the Reynolds operator is the first return value and in case of characteristic 0 a 1x2 <matrix> giving enumerator and denominator of Molien series is the second one; in case of prime characteristic a ring with the name newring of characteristic 0 is created where the same Molien series (named M) is stored

Display: information if the third component of flags does not equal 0

Theory: The entire matrix group is generated by getting all left products of the generators with new elements from the last run through the loop (or the generators themselves during the first run). All the ones that have been generated before are thrown out and the program terminates when are no new elements found in one run. Additionally each time a new group element is found the corresponding ring mapping of which the Reynolds operator is made up is generated. They are stored in the rows of the first return value. In characteristic 0 the terms 1/det(1-xE) is computed whenever a new element E is found. In prime characteristic a Brauer lift is involved and the terms are only computed after the entire matrix group is generated (to avoid the modular case). The returned matrix gives enumerator and denominator of the expanded version where common factors have been canceled.

Example:

```
LIB "finvar.lib";
"         note the case of prime characteristic";
↦          note the case of prime characteristic
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY,M=reynolds_molien(A);
print(REY);
↦ y, -x,-z,
↦ -x,-y,z,
↦ -y,x, -z,
↦ x, y, z
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
ring S=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY=reynolds_molien(A,"");
print(REY);
↦ y, -x,-z,
↦ -x,-y,z,
↦ -y,x, -z,
↦ x, y, z
setring Finvar::newring;
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
setring S;
kill Finvar::newring;
```

D.7.1.11 partial_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: partial_molien(M,n[,p]);
M: a 1x2 <matrix>, n: an <int> indicating number of terms in the expansion, p: an optional <poly>

Assume: M is the return value of molien or the second return value of reynolds_molien, p ought to be the second return value of a previous run of partial_molien and avoids recalculating known terms

Return: n terms (type <poly>) of the partial expansion of the Molien series (first n if there is no third parameter given, otherwise the next n terms depending on a previous calculation) and an intermediate result (type <poly>) of the calculation to be used as third parameter in a next run of partial_molien

Theory: The following calculation is implemented:

(1+a1x+a2x^2+...+anx^n)/(1+b1x+b2x^2+...+bmx^m)=(1+(a1-b1)x+...
(1+b1x+b2x^2+...+bmx^m)

___

(a1-b1)x+(a2-b2)x^2+...
(a1-b1)x+b1(a1-b1)x^2+...

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY,M=reynolds_molien(A);
poly p(1..2);
p(1..2)=partial_molien(M,5);
p(1);
↦ 4x5+5x4+2x3+2x2+1
```

```
p(1..2)=partial_molien(M,5,p(2));
p(1);
↦ 18x10+12x9+13x8+8x7+8x6
```

D.7.1.12 evaluate_reynolds

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: evaluate_reynolds(REY,I);
REY: a <matrix> representing the Reynolds operator, I: an arbitrary <ideal>

Assume: REY is the first return value of group_reynolds() or reynolds_molien()

Returns: image of the polynomials defining I under the Reynolds operator (type <ideal>)

Note: the characteristic of the coefficient field of the polynomial ring should not divide the order of the finite matrix group

Theory: REY has been constructed in such a way that each row serves as a ring mapping of which the Reynolds operator is made up.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
ideal I=x2,y2,z2;
print(evaluate_reynolds(L[1],I));
↦ 1/2x2+1/2y2,
↦ 1/2x2+1/2y2,
↦ z2
```

D.7.1.13 invariant_basis

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_basis(g,G1,G2,...);
g: an <int> indicating of which degree (>0) the homogeneous basis should be, G1,G2,...: <matrices> generating a finite matrix group

Returns: the basis (type <ideal>) of the space of invariants of degree g

Theory: A general polynomial of degree g is generated and the generators of the matrix group applied. The difference ought to be 0 and this way a system of linear equations is created. It is solved by computing syzygies.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
print(invariant_basis(2,A));
↦ x2+y2,
↦ z2
```

D.7.1.14 invariant_basis_reynolds

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: invariant_basis_reynolds(REY,d[,flags]);
REY: a <matrix> representing the Reynolds operator, d: an <int> indicating of which degree (>0) the homogeneous basis should be, flags: an optional <intvec> with two entries: its first component gives the dimension of the space (default <0 meaning unknown) and its second component is used as the number of polynomials that should be mapped to invariants during one call of evaluate_reynolds if the dimension of the space is unknown or the number such that number x dimension polynomials are mapped to invariants during one call of evaluate_reynolds

Assume: REY is the first return value of group_reynolds() or reynolds_molien() and flags[1] given by partial_molien

Return: the basis (type <ideal>) of the space of invariants of degree d

Theory: Monomials of degree d are mapped to invariants with the Reynolds operator. A linearly independent set is generated with the help of minbase.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
intvec flags=0,1,0;
matrix REY,M=reynolds_molien(A,flags);
flags=8,6;
print(invariant_basis_reynolds(REY,6,flags));
↦ z6,
↦ x2z4+y2z4,
↦ x2y2z2,
↦ x3yz2-xy3z2,
↦ x4z2+y4z2,
↦ x4y2+x2y4,
↦ x5y-xy5,
↦ x6+y6
```

D.7.1.15 primary_char0

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_char0(REY,M[,v]);
REY: a <matrix> representing the Reynolds operator, M: a 1x2 <matrix> representing the Molien series, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien and M the one of molien or the second one of reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY,M=reynolds_molien(A);
matrix P=primary_char0(REY,M);
print(P);
↦ z2,x2+y2,x2y2
```

D.7.1.16 primary_charp

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp(REY,ringname[,v]);
REY: a <matrix> representing the Reynolds operator,
Finvar::newring: the ring where the Molien series is stored, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien and ringname gives the name of a ring of characteristic 0 that has been created by molien or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
molien(L[2..size(L)],"");
matrix P=primary_charp(L[1],"");
kill Finvar::newring;
print(P);
↦ z2,x2+y2,x2y2
```

D.7.1.17 primary_char0_no_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_char0_no_molien(REY[,v]);
REY: a <matrix> representing the Reynolds operator, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and an <intvec> listing some of the degrees where no non-trivial homogeneous invariants are to be found

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
list l=primary_char0_no_molien(L[1]);
print(l[1]);
↦ z2,x2+y2,x2y2
```

D.7.1.18 primary_charp_no_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp_no_molien(REY[,v]);
REY: a <matrix> representing the Reynolds operator, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and an <intvec> listing some of the degrees where no non-trivial homogeneous invariants are to be found

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
list l=primary_charp_no_molien(L[1]);
print(l[1]);
↦ z2,x2+y2,x2y2
```

D.7.1.19 primary_charp_without

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp_without(G1,G2,...[,v]);
G1,G2,...: <matrices> generating a finite matrix group, v: an optional <int>

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and those are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see paper "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)). No Reynolds operator or Molien series is used.

Example:

```
LIB "finvar.lib";
ring R=2,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix P=primary_charp_without(A);
print(P);
↦ x+y,z,xy
```

D.7.1.20 primary_char0_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_char0_random(REY,M,r[,v]);
REY: a <matrix> representing the Reynolds operator, M: a 1x2 <matrix> representing the Molien series, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien and M the one of molien or the second one of reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY,M=reynolds_molien(A);
matrix P=primary_char0_random(REY,M,1);
print(P);
↦ z2,x2+y2,x4+y4-z4
```

D.7.1.21 primary_charp_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp_random(REY,ringname,r[,v]);
REY: a <matrix> representing the Reynolds operator,
Finvar::newring: the ring where the Molien series is stored, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien and ringname gives the name of a ring of characteristic 0 that has been created by molien or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
molien(L[2..size(L)],"");
matrix P=primary_charp_random(L[1],"",1);
kill Finvar::newring;
print(P);
↦ z2,x2+y2,x4+y4-z4
```

D.7.1.22 primary_char0_no_molien_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_char0_no_molien_random(REY,r[,v]);
REY: a <matrix> representing the Reynolds operator, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and an <intvec> listing some of the degrees where no non-trivial homogeneous invariants are to be found

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
list l=primary_char0_no_molien_random(L[1],1);
print(l[1]);
↦ z2,x2+y2,x4+y4-z4
```

D.7.1.23 primary_charp_no_molien_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp_no_molien_random(REY,r[,v]);
REY: a <matrix> representing the Reynolds operator, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, v: an optional <int>

Assume: REY is the first return value of group_reynolds or reynolds_molien

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring and an <intvec> listing some of the degrees where no non-trivial homogeneous invariants are to be found

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)).

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
list l=primary_charp_no_molien_random(L[1],1);
print(l[1]);
↦ z2,x2+y2,x4+y4-z4
```

D.7.1.24 primary_charp_without_random

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: primary_charp_without_random(G1,G2,...,r[,v]);
G1,G2,...: <matrices> generating a finite matrix group, r: an <int> where -|r| to |r| is the range of coefficients of the random combinations of bases elements, v: an optional <int>

Display: information about the various stages of the programme if v does not equal 0

Return: primary invariants (type <matrix>) of the invariant ring

Theory: Bases of homogeneous invariants are generated successively and random linear combinations are chosen as primary invariants that lower the dimension of the ideal generated by the previously found invariants (see "Generating a Noetherian Normalization of the Invariant Ring of a Finite Group" by Decker, Heydtmann, Schreyer (1998)). No Reynolds operator or Molien series is used.

Example:

```
LIB "finvar.lib";
ring R=2,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix P=primary_charp_without_random(A,1);
print(P);
↦ x+y,z,xy
```

D.7.1.25 power_products

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: power_products(dv,d);
dv: an <intvec> giving the degrees of homogeneous polynomials, d: the degree of the desired power products

Return: a size(dv)*m <intmat> where each column ought to be interpreted as containing the exponents of the corresponding polynomials. The product of the powers is then homogeneous of degree d.

Example:

```
LIB "finvar.lib";
intvec dv=5,5,5,10,10;
print(power_products(dv,10));
↦      2      1      1      0      0      0      0      0
↦      0      1      0      2      1      0      0      0
↦      0      0      1      0      1      2      0      0
↦      0      0      0      0      0      0      1      0
↦      0      0      0      0      0      0      0      1
print(power_products(dv,7));
↦      0
↦      0
↦      0
↦      0
↦      0
```

D.7.1.26 secondary_char0

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: secondary_char0(P,REY,M[,v][,"old"]);
P: a 1xn <matrix> with homogeneous primary invariants, where n is the number of variables of the basering;
REY: a gxn <matrix> representing the Reynolds operator, where g the size of the corresponding group;
M: a 1x2 <matrix> giving numerator and denominator of the Molien series;
v: an optional <int>;
"old": if this string occurs as (optional) parameter, then an old version of secondary_char0 is used (for downward
compatibility)

Assume: The characteristic of basering is zero;
REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants();
M is the return value of molien() or the second one of reynolds_molien() or the third one of primary_invariants()

Return: Homogeneous secondary invariants and irreducible secondary invariants of the invariant ring (both type <matrix>)

Display: Information on the progress of the computations if v is an integer different from 0.

Theory: The secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo the primary invariants, mapping those to invariants with the Reynolds operator. Among these images or their power products we pick secondary invariants using Groebner basis techniques (see S. King: Fast Computation of Secondary Invariants).
The size of this set can be read off from the Molien series.

Note: Secondary invariants are not uniquely determined by the given data. Specifically, the output of secondary_char0(P,REY,M,"old") will differ from the output of secondary_char0(P,REY,M). However, the ideal generated by the irreducible homogeneous secondary invariants will be the same in both cases.
There are three internal parameters "pieces", "MonStep" and "IrrSwitch". The default values of the parameters should be fine in most cases. However, in some cases,

different values may provide a better balance of memory consumption (smaller values) and speed (bigger values).

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A);
matrix S,IS=secondary_char0(L[1..3],1);
↦
↦ We need to find
↦ 1 secondary invariant in degree 0
↦ 0 secondary invariants in degree 1
↦ 0 secondary invariants in degree 2
↦ 2 secondary invariants in degree 3
↦ 1 secondary invariant in degree 4
↦ In degree 0 we have: 1
↦
↦ Searching in degree  3 , we need to find  2  invariant(s)...
↦   Looking for Power Products...
↦   There are  2 irreducible secondary invariants in degree  3
↦     We found all 2 irreducibles in degree  3
↦
↦ Searching in degree  4 , we need to find  1  invariant(s)...
↦   Looking for Power Products...
↦   There are  1 irreducible secondary invariants in degree  4
↦     We found all 1 irreducibles in degree  4
↦
↦   We're done!
↦
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
print(IS);
↦ xyz,x2z-y2z,x3y-xy3
```

See also: Section D.7.1.27 [irred_secondary_char0], page 1792.

D.7.1.27 irred_secondary_char0

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: irred_secondary_char0(P,REY,M[,v][,"PP"]);
P: a 1xn <matrix> with homogeneous primary invariants, where n is the number of variables of the basering;
REY: a gxn <matrix> representing the Reynolds operator, where g the size of the corresponding group;
M: a 1x2 <matrix> giving numerator and denominator of the Molien series;
v: an optional <int>;
"PP": if this string occurs as (optional) parameter, then in all degrees power products of irr. sec. inv. will be computed.

Return: Irreducible homogeneous secondary invariants of the invariant ring (type <matrix>)

Assume: We are in the non-modular case, i.e., the characteristic of the basering does not divide the group order;

REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants();
M is the return value of molien() or the second one of reynolds_molien() or the third one of primary_invariants()

Display: Information on the progress of computations if v does not equal 0

Theory: The secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo the primary invariants, mapping those to invariants with the Reynolds operator. Among these images or their power products we pick secondary invariants using Groebner basis techniques (see S. King: Fast Computation of Secondary Invariants). The size of this set can be read off from the Molien series. Here, only irreducible secondary invariants are explicitly computed, which saves time and memory.
Moreover, if no irr. sec. inv. in degree d-1 have been found and unless the last optional parameter "PP" is used, a Groebner basis of primary invariants and irreducible secondary invariants up to degree d-2 is computed, which allows to detect irr. sec. inv. in degree d without computing power products.
There are three internal parameters "pieces", "MonStep" and "IrrSwitch". The default values of the parameters should be fine in most cases. However, in some cases, different values may provide a better balance of memory consumption (smaller values) and speed (bigger values).

Example:

```
LIB "finvar.lib";
ring r= 0, (a,b,c,d,e,f),dp;
matrix A1[6][6] = 0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
matrix A2[6][6] = 0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,
list L = primary_invariants(A1,A2);
matrix IS = irred_secondary_char0(L[1],L[2],L[3],0);
IS;
↦ IS[1,1]=ab+cd+ef
↦ IS[1,2]=a2d+ad2+b2e+be2+c2f+cf2
↦ IS[1,3]=ac2+b2d+a2e+ce2+d2f+bf2
↦ IS[1,4]=b2c+bc2+d2e+de2+a2f+af2
↦ IS[1,5]=a2c+bd2+c2e+ae2+b2f+df2
↦ IS[1,6]=a2cd+abd2+abe2+b2ef+c2ef+cdf2
↦ IS[1,7]=ab2d+ac2d+a2be+cd2f+ce2f+bef2
↦ IS[1,8]=a2bd+acd2+ab2e+c2df+be2f+cef2
↦ IS[1,9]=a3d+ad3+b3e+be3+c3f+cf3
```

See also: Section D.7.1.26 [secondary_char0], page 1791.

D.7.1.28 secondary_charp

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: secondary_charp(P,REY,ringname[,v][,"old"]);
P: a 1xn <matrix> with homogeneous primary invariants, where n is the number of variables of the basering;
REY: a gxn <matrix> representing the Reynolds operator, where g the size of the corresponding group;
Finvar::newring is the ring of characteristic 0 containing a 1x2 <matrix> M giving numerator and denominator of the Molien series;
v: an optional <int>;

"old": if this string occurs as (optional) parameter, then an old version of secondary_char0 is used (for downward
compatibility)

Assume: The characteristic of basering is not zero;
REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants();
Finvar::newring is the ring of characteristic 0 that has been created by molien() or reynolds_molien() or primary_invariants()

Return: secondary invariants of the invariant ring (type <matrix>) and irreducible secondary invariants (type <matrix>)

Display: information if v does not equal 0

Theory: The secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo the primary invariants, mapping those to invariants with the Reynolds operator. Among these images or their power products we pick secondary invariants using Groebner basis techniques (see S. King: Fast Computation of Secondary Invariants). The size of this set can be read off from the Molien series.

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A);
matrix S,IS=secondary_charp(L[1..size(L)],1);
↦
↦ We need to find
↦ 1 secondary invariant in degree 0
↦ 0 secondary invariants in degree 1
↦ 0 secondary invariants in degree 2
↦ 2 secondary invariants in degree 3
↦ 1 secondary invariant in degree 4
↦ In degree 0 we have: 1
↦
↦ Searching in degree  3 , we need to find  2  invariant(s)...
↦   Looking for Power Products...
↦   There are  2 irreducible secondary invariants in degree  3
↦     We found all 2 irreducibles in degree  3
↦
↦ Searching in degree  4 , we need to find  1  invariant(s)...
↦   Looking for Power Products...
↦   There are  1 irreducible secondary invariants in degree  4
↦     We found all 1 irreducibles in degree  4
↦
↦   We're done!
↦
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
print(IS);
↦ xyz,x2z-y2z,x3y-xy3
```

D.7.1.29 secondary_no_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: secondary_no_molien(P,REY[,deg_vec,v]);
P: a 1xn <matrix> with primary invariants, REY: a gxn <matrix> representing the Reynolds operator, deg_vec: an optional <intvec> listing some degrees where no non-trivial homogeneous invariants can be found, v: an optional <int>

Assume: n is the number of variables of the basering, g the size of the group, REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants(), deg_vec is the second return value of primary_char0_no_molien(), primary_charp_no_molien(), primary_char0_no_molien_random() or primary_charp_no_molien_random()

Return: secondary invariants of the invariant ring (type <matrix>)

Display: information if v does not equal 0

Theory: Secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo primary invariants, mapping those to invariants with the Reynolds operator and using these images as candidates for secondary invariants. We have the Reynolds operator, hence, we are in the non-modular case. Therefore, the invariant ring is Cohen-Macaulay, hence the number of secondary invariants is the product of the degrees of primary invariants divided by the group order.

Note: <secondary_and_irreducibles_no_molien> should usually be faster and of more useful functionality.

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A,intvec(1,1,0));
// In that example, there are no secondary invariants
// in degree 1 or 2.
matrix S=secondary_no_molien(L[1..2],intvec(1,2),1);
↦   We need to find  4  secondary invariants.
↦
↦ In degree 0 we have: 1
↦
↦ Searching in degree  3 ...
↦     We found sec. inv. number  2  in degree  3
↦     We found sec. inv. number  3  in degree  3
↦ Searching in degree  4 ...
↦     We found sec. inv. number  4  in degree  4
↦
↦   We're done!
↦
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
```

See also: Section D.7.1.31 [secondary_and_irreducibles_no_molien], page 1796.

D.7.1.30 irred_secondary_no_molien

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: irred_secondary_no_molien(P,REY[,deg_vec,v]);
P: a 1xn <matrix> with primary invariants, REY: a gxn <matrix> representing the

Reynolds operator, deg_vec: an optional <intvec> listing some degrees where no irreducible secondary invariants can be found, v: an optional <int>

Assume: n is the number of variables of the basering, g the size of the group, REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants()

Return: Irreducible secondary invariants of the invariant ring (type <matrix>)

Display: information if v does not equal 0

Theory: Irred. secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo primary and previously found secondary invariants, mapping those to invariants with the Reynolds operator. Among these images we pick secondary invariants, using Groebner basis techniques.

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A,intvec(1,1,0));
// In that example, there are no secondary invariants
// in degree 1 or 2.
matrix IS=irred_secondary_no_molien(L[1..2],intvec(1,2),1);
↦
↦ Searching irred. sec. inv. in degree  3
↦   We have  4  candidates for irred. secondaries
↦     We found irr. sec. inv. number  1  in degree  3
↦     We found irr. sec. inv. number  2  in degree  3
↦ Searching irred. sec. inv. in degree  4
↦   We have  1  candidates for irred. secondaries
↦     We found irr. sec. inv. number  3  in degree  4
↦ Searching irred. sec. inv. in degree  5
↦ Searching irred. sec. inv. in degree  6
↦ Searching irred. sec. inv. in degree  7
↦ Searching irred. sec. inv. in degree  8
↦ Searching irred. sec. inv. in degree  9
↦ Searching irred. sec. inv. in degree  10
↦ Searching irred. sec. inv. in degree  11
↦ Searching irred. sec. inv. in degree  12
↦ Searching irred. sec. inv. in degree  13
print(IS);
↦ x2z-y2z,xyz,x3y-xy3
```

See also: Section D.7.1.27 [irred_secondary_char0], page 1792.

D.7.1.31 secondary_and_irreducibles_no_molien

Procedure from library finvar.lib (see Section D.7.1 [finvar_lib], page 1774).

Usage: secondary_and_irreducibles_no_molien(P,REY[,deg_vec,v]); P: a 1xn <matrix> with primary invariants, REY: a gxn <matrix> representing the Reynolds operator, deg_vec: an optional <intvec> listing some degrees where no non-trivial homogeneous invariants can be found, v: an optional <int>

Assume: n is the number of variables of the basering, g the size of the group, REY is the 1st return value of group_reynolds(), reynolds_molien() or the second one of primary_invariants()

Return: secondary invariants of the invariant ring (type <matrix>) and irreducible secondary invariants (type <matrix>)

Display: information if v does not equal 0

Theory: Secondary invariants are calculated by finding a basis (in terms of monomials) of the basering modulo primary invariants, mapping those to invariants with the Reynolds operator. Among these images or their power products we pick secondary invariants using Groebner basis techniques (see S. King: Fast Computation of Secondary Invariants). We have the Reynolds operator, hence, we are in the non-modular case. Therefore, the invariant ring is Cohen-Macaulay, hence the number of secondary invariants is the product of the degrees of primary invariants divided by the group order.

Example:

```
LIB "finvar.lib";
ring R=3,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A,intvec(1,1,0));
// In that example, there are no secondary invariants
// in degree 1 or 2.
matrix S,IS=secondary_and_irreducibles_no_molien(L[1..2],intvec(1,2),1);
↦
↦   We need to find  4  secondary invariants.
↦
↦ In degree 0 we have: 1
↦
↦ Searching in degree  3
↦   Looking for Power Products...
↦   Looking for irreducible secondary invariants in degree  3
↦     We found irreducible sec. inv. number 1 in degree 3
↦     We found irreducible sec. inv. number 2 in degree 3
↦
↦ Searching in degree  4
↦   Looking for Power Products...
↦   Looking for irreducible secondary invariants in degree  4
↦     We found irreducible sec. inv. number 1 in degree 4
↦
↦
↦   We're done!
↦
print(S);
↦ 1,xyz,x2z-y2z,x3y-xy3
print(IS);
↦ xyz,x2z-y2z,x3y-xy3
```

See also: Section D.7.1.29 [secondary_no_molien], page 1794.

D.7.1.32 secondary_not_cohen_macaulay

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: secondary_not_cohen_macaulay(P,G1,G2,...[,v]);
P: a 1xn <matrix> with primary invariants, G1,G2,...: nxn <matrices> generating a finite matrix group, v: optional <int>

Assume: n is the number of variables of the basering

Return: secondary invariants of the invariant ring (type <matrix>)

Display: information on the progress of computation if v does not equal 0

Theory: Secondary invariants are generated following "Generating Invariant Rings of Finite Groups over Arbitrary Fields" by Kemper (1996).

Example:

```
LIB "finvar.lib";
ring R=2,(x,y,z),dp;
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=primary_invariants(A);
matrix S=secondary_not_cohen_macaulay(L[1],A);
print(S);
↦ 1
```

D.7.1.33 orbit_variety

Procedure from library finvar.lib (see Section D.7.1 [finvar_lib], page 1774).

Usage: orbit_variety(F,s);
F: a 1xm <matrix> defing an invariant ring, s: a <string> giving the name for a new ring

Return: a Groebner basis (type <ideal>, named G) for the ideal defining the orbit variety (i.e. the syzygy ideal) in the new ring (named newring)

Theory: The ideal of algebraic relations of the invariant ring generators is calculated, then the variables of the original ring are eliminated and the polynomials that are left over define the orbit variety

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix F[1][7]=x2+y2,z2,x4+y4,1,x2z-1y2z,xyz,x3y-1xy3;
orbit_variety(F,"");
print(G);
↦ y(4)-1,
↦ y(5)*y(6)-y(2)*y(7),
↦ y(2)*y(3)-y(5)^2-2*y(6)^2,
↦ y(1)^2*y(6)-2*y(3)*y(6)+y(5)*y(7),
↦ y(1)^2*y(5)-y(3)*y(5)-2*y(6)*y(7),
↦ y(1)^2*y(2)-y(2)*y(3)-2*y(6)^2,
↦ y(1)^4-3*y(1)^2*y(3)+2*y(3)^2+2*y(7)^2
basering;
↦ // coefficients: QQ
↦ // number of vars : 7
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2) y(3) y(4) y(5) y(6) y(7)
↦ //        block   2 : ordering C
```

D.7.1.34 rel_orbit_variety

Procedure from library finvar.lib (see Section D.7.1 [finvar_lib], page 1774).

Usage: rel_orbit_variety(I,F[,s]);
I: an <ideal> invariant under the action of a group,
F: a 1xm <matrix> defining the invariant ring of this group.
s: optional <string>; if s is present then (for downward compatibility) the old procedure <relative_orbit_variety> is called, and in this case s gives the name of a new <ring>.

Return: Without optional string s, a list L of two rings is returned.
The ring L[1] carries a weighted degree order with variables y(1..m), the weight of y(k) equal to the degree of the k-th generators F[1,k] of the invariant ring.
L[1] contains a Groebner basis (type <ideal>, named G) of the ideal defining the relative orbit variety with respect to I.
The ring L[2] has the variables of the basering together with y(1..m) and carries a block order: The first block is the order of the basering, the second is the weighted degree order occurring in L[1]. L[2] contains G and a Groebner basis (type <ideal>, named Conv) such that if p is any invariant polynomial expressed in the variables of the basering then reduce(p,Conv) is a polynomial in the new variables y(1..m) such that evaluation at the generators of the invariant ring yields p. This can be used to avoid the application of <algebra_containment> (see Section D.4.2.1 [algebra_containment], page 1004).
For the case of optional string s, the function is equivalent to Section D.7.1.35 [relative_orbit_variety], page 1800.

Theory: A Groebner basis of the ideal of algebraic relations of the invariant ring generators is calculated, then one of the basis elements plus the ideal generators. The variables of the original ring are eliminated and the polynomials that are left define the relative orbit variety with respect to I. The elimination is done by a weighted blockorder that has the advantage of dealing with quasi-homogeneous ideals.

Note: We provide the ring L[1] for the sake of downward compatibility, since it is closer to the ring returned by relative_orbit_variety than L[2]. However, L[1] carries a weighted degree order, whereas the ring returned by relative_orbit_variety is lexicographically ordered.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix F[1][3]=x+y+z,xy+xz+yz,xyz;
ideal I=x2+y2+z2-1,x2y+y2z+z2x-2x-2y-2z,xy2+yz2+zx2-2x-2y-2z;
list L = rel_orbit_variety(I,F);
↦
↦ // 'rel_orbit_variety' created a list of two rings.
↦ // If L is the name of that list, you can access
↦ // the first ring by
↦ //   def R = L[1]; setring R;
↦ // (similarly for the second ring)
def AllR = L[2];
setring(AllR);
print(G);
↦ y(1)^2-2*y(2)-1,
↦ y(1)*y(2)-3*y(3)-4*y(1),
↦ 2*y(2)^2-3*y(1)*y(3)-7*y(2)-4,
↦ 6*y(3)^2-15*y(1)*y(3)+25*y(2)+12
print(Conv);
↦ x+y+z-y(1),
↦ y^2+y*z+z^2-y*y(1)-z*y(1)+y(2),
↦ z^3-z^2*y(1)+z*y(2)-y(3)
basering;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering wp
↦ //                  : names    y(1) y(2) y(3)
↦ //                  : weights     1    2    3
↦ //        block   3 : ordering C
```

See also: Section D.7.1.35 [relative_orbit_variety], page 1800.

D.7.1.35 relative_orbit_variety

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: relative_orbit_variety(I,F,s);
I: an <ideal> invariant under the action of a group,
F: a 1xm <matrix> defining the invariant ring of this group,
Finvar::newring: the new ring

Return: The procedure ends with a new ring named newring.
It contains a Groebner basis
(type <ideal>, named G) for the ideal defining the
relative orbit variety with respect to I in the new ring.

Theory: A Groebner basis of the ideal of algebraic relations of the invariant ring generators is calculated, then one of the basis elements plus the ideal generators. The variables of the original ring are eliminated and the polynomials that are left define the relative orbit variety with respect to I.

Note: This procedure is now replaced by rel_orbit_variety (see Section D.7.1.34 [rel_orbit_variety], page 1798), which uses a different elemination order that should usually allow faster computations.

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix F[1][3]=x+y+z,xy+xz+yz,xyz;
ideal I=x2+y2+z2-1,x2y+y2z+z2x-2x-2y-2z,xy2+yz2+zx2-2x-2y-2z;
string newring="E";
relative_orbit_variety(I,F,newring);
print(G);
↦ 27*y(3)^6-513*y(3)^4+33849*y(3)^2-784,
↦ 1475*y(2)+9*y(3)^4-264*y(3)^2+736,
↦ 8260*y(1)+9*y(3)^5-87*y(3)^3+5515*y(3)
basering;
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering lp
↦ //                  : names    y(1) y(2) y(3)
↦ //        block   2 : ordering C
```

See also: Section D.7.1.34 [rel_orbit_variety], page 1798.

D.7.1.36 image_of_variety

Procedure from library `finvar.lib` (see Section D.7.1 [finvar_lib], page 1774).

Usage: image_of_variety(I,F);
I: an arbitrary <ideal>,
F: a 1xm <matrix> defining an invariant ring of some matrix group

Return: The <ideal> defining the image under that group of the variety defined by I

Theory: rel_orbit_variety(I,F) is called and the newly introduced variables in the output are replaced by the generators of the invariant ring. This ideal in the original variables defines the image of the variety defined by I

Example:

```
LIB "finvar.lib";
ring R=0,(x,y,z),dp;
matrix F[1][3]=x+y+z,xy+xz+yz,xyz;
ideal I=xy;
print(image_of_variety(I,F));
↦ xyz
```

D.7.2 ainvar_lib

Library: ainvar.lib

Purpose: Invariant Rings of the Additive Group

Authors: Gerhard Pfister (email: pfister@mathematik.uni-kl.de), Gert-Martin Greuel (email: greuel@mathematik.uni-kl.de)

Procedures:

D.7.2.1 invariantRing

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: invariantRing(m,p,q,b[,r,pa]); m matrix, p,q poly, b,r int, pa string

Assume: p,q variables with m(p)=q and q invariant under m
i.e. if p=x(i) and q=x(j) then m[j,1]=0 and m[i,1]=x(j)

Return: ideal, containing generators of the ring of invariants of the additive group (K,+) given by the vector field

m = m[1,1]*d/dx(1) +...+ m[n,1]*d/dx(n).

If b>0 the computation stops after all invariants of degree <= b (and at least one of higher degree) are found or when all invariants are computed.
If b<=0, the computation continues until all generators of the ring of invariants are computed (should be used only if the ring of invariants is known to be finitely generated, otherwise the algorithm might not stop).
If r=1 a different reduction is used which is sometimes faster (default r=0).

Display: if pa is given (any string as 5th or 6th argument), the computation pauses whenever new invariants are found and displays them

Theory: The algorithm for computing the ring of invariants works in char 0 or suffiently large characteristic.
(K,+) acts as the exponential of the vector field defined by the matrix m.
For background see G.-M. Greuel, G. Pfister,
Geometric quotients of unipotent group actions, Proc. London Math. Soc. (3) 67, 75-105 (1993).

Example:

```
LIB "ainvar.lib";
//Winkelmann: free action but Spec(k[x(1),...,x(5)]) --> Spec(invariant ring)
//is not surjective
ring rw=0,(x(1..5)),dp;
matrix m[5][1];
m[3,1]=x(1);
m[4,1]=x(2);
m[5,1]=1+x(1)*x(4)+x(2)*x(3);
ideal in=invariantRing(m,x(3),x(1),0);      //compute full invarint ring
in;
↦ in[1]=x(1)
↦ in[2]=x(2)
↦ in[3]=x(2)*x(3)*x(4)-x(2)*x(5)+x(4)
↦ in[4]=x(1)*x(3)*x(4)-x(1)*x(5)+x(3)
//Deveney/Finston: The ring of invariants is not finitely generated
ring rf=0,(x(1..7)),dp;
matrix m[7][1];
m[4,1]=x(1)^3;
m[5,1]=x(2)^3;
m[6,1]=x(3)^3;
m[7,1]=(x(1)*x(2)*x(3))^2;
ideal in=invariantRing(m,x(4),x(1),6);      //all invariants up to degree 6
in;
↦ in[1]=x(1)
↦ in[2]=x(3)
↦ in[3]=x(2)
↦ in[4]=x(3)^3*x(4)-x(1)^3*x(6)
↦ in[5]=x(2)^3*x(4)-x(1)^3*x(5)
↦ in[6]=x(2)^2*x(3)^2*x(4)-x(1)*x(7)
↦ in[7]=x(1)^2*x(2)^2*x(6)-x(3)*x(7)
↦ in[8]=x(1)^2*x(3)^2*x(5)-x(2)*x(7)
↦ in[9]=x(1)^2*x(2)*x(3)^4*x(4)*x(5)+x(1)^2*x(2)^4*x(3)*x(4)*x(6)-x(1)^5*x(\
   2)*x(3)*x(5)*x(6)-2*x(2)^2*x(3)^2*x(4)*x(7)+x(1)*x(7)^2
```

D.7.2.2 derivate

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: derivate(m,id); m matrix, id poly/vector/ideal

Assume: m is an nx1 matrix, where n = number of variables of the basering

Return: poly/vector/ideal (same type as input), result of applying the vector field by the matrix m componentwise to id;

Note: the vector field is m[1,1]*d/dx(1) +...+ m[1,n]*d/dx(n)

Example:

```
LIB "ainvar.lib";
ring q=0,(x,y,z,u,v,w),dp;
poly f=2xz-y2;
matrix m[6][1] =x,y,0,u,v;
derivate(m,f);
↦ -2y2+2xz
vector v = [2xz-y2,u6-3];
derivate(m,v);
↦ 6u6*gen(2)-2y2*gen(1)+2xz*gen(1)
derivate(m,ideal(2xz-y2,u6-3));
↦ _[1]=-2y2+2xz
↦ _[2]=6u6
```

D.7.2.3 actionIsProper

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: actionIsProper(m); m matrix

Assume: m is a nx1 matrix, where n = number of variables of the basering

Return: int = 1, if the action defined by m is proper, 0 if not

Note: m defines a group action which is the exponential of the vector field m[1,1]*d/dx(1) +...+ m[1,n]*d/dx(n)

Example:

```
LIB "ainvar.lib";
ring rf=0,x(1..7),dp;
matrix m[7][1];
m[4,1]=x(1)^3;
m[5,1]=x(2)^3;
m[6,1]=x(3)^3;
m[7,1]=(x(1)*x(2)*x(3))^2;
actionIsProper(m);
↦ 0
ring rd=0,x(1..5),dp;
matrix m[5][1];
m[3,1]=x(1);
m[4,1]=x(2);
m[5,1]=1+x(1)*x(4)^2;
actionIsProper(m);
↦ 1
```

D.7.2.4 reduction

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: reduction(p,I[,q,n]); p poly, I ideal, [q monomial, n int (optional)]

Return: a polynomial equal to p-H(f1,...,fr), in case the leading term LT(p) of p is of the form H(LT(f1),...,LT(fr)) for some polynomial H in r variables over the base field, I=f1,...,fr; if q is given, a maximal power a is computed such that q^a divides p-H(f1,...,fr), and then (p-H(f1,...,fr))/q^a is returned; return p if no H is found
if n=1, a different algorithm is chosen which is sometimes faster (default: n=0; q and n can be given (or not) in any order)

Note: this is a kind of SAGBI reduction in the subalgebra K[f1,...,fr] of the basering

Example:

```
LIB "ainvar.lib";
ring q=0,(x,y,z,u,v,w),dp;
poly p=x2yz-x2v;
ideal dom =x-w,u2w+1,yz-v;
reduction(p,dom);
↦ 2xyzw-yzw2-2xvw+vw2
reduction(p,dom,w);
↦ 2xyz-yzw-2xv+vw
```

D.7.2.5 completeReduction

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: completeReduction(p,I[,q,n]); p poly, I ideal, [q monomial, n int]

Return: a polynomial, the SAGBI reduction of the polynomial p with respect to I via the procedure 'reduction' as long as possible
if n=1, a different algorithm is chosen which is sometimes faster (default: n=0; q and n can be given (or not) in any order)

Note: help reduction; shows an explanation of SAGBI reduction

Example:

```
LIB "ainvar.lib";
ring q=0,(x,y,z,u,v,w),dp;
poly p=x2yz-x2v;
ideal dom =x-w,u2w+1,yz-v;
completeReduction(p,dom);
↦ 2xyzw-yzw2-2xvw+vw2
completeReduction(p,dom,w);
↦ 0
```

D.7.2.6 localInvar

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: localInvar(m,p,q,h); m matrix, p,q,h polynomials

Assume: m(q) and h are invariant under the vector field m, i.e. m(m(q))=m(h)=0 h must be a ring variable

Return: a polynomial, the invariant polynomial of the vector field

m = m[1,1]*d/dx(1) +...+ m[n,1]*d/dx(n)

with respect to p,q,h. It is defined as follows: set inv = p if p is invariant, and else set inv = m(q)^N * sum_i=1..N-1{ (-1)^i*(1/i!)*m^i(p)*(q/m(q))^i } where m^N(p) = 0, m^(N-1)(p) != 0; the result is inv divided by h as often as possible

Example:

```
LIB "ainvar.lib";
ring q=0,(x,y,z),dp;
matrix m[3][1];
m[2,1]=x;
m[3,1]=y;
```

```
poly in=localInvar(m,z,y,x);
in;
↦ -1/2y2+xz
```

D.7.2.7 furtherInvar

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: furtherInvar(m,id,karl,q); m matrix, id,karl ideals, q poly, n int

Assume: karl,id,q are invariant under the vector field m,
moreover, q must be a variable

Return: list of two ideals, the first ideal contains further invariants of the vector field

m = sum m[i,1]*d/dx(i) with respect to id,p,q,

i.e. we compute elements in the (invariant) subring generated by id which are divisible by q and divide them by q as often as possible. The second ideal contains all invariants given before. If n=1, a different algorithm is chosen which is sometimes faster (default: n=0)

Example:

```
LIB "ainvar.lib";
ring r=0,(x,y,z,u),dp;
matrix m[4][1];
m[2,1]=x;
m[3,1]=y;
m[4,1]=z;
ideal id=localInvar(m,z,y,x),localInvar(m,u,y,x);
ideal karl=id,x;
list in=furtherInvar(m,id,karl,x);
in;
↦ [1]:
↦    _[1]=y2z2-8/3xz3-2y3u+6xyzu-3x2u2
↦ [2]:
↦    _[1]=-1/2y2+xz
↦    _[2]=1/3y3-xyz+x2u
↦    _[3]=x
```

D.7.2.8 sortier

Procedure from library `ainvar.lib` (see Section D.7.2 [ainvar_lib], page 1801).

Usage: sortier(id); id ideal/module

Return: the same ideal/module but with generators ordered by their leading terms, starting with the smallest

Example:

```
LIB "ainvar.lib";
ring q=0,(x,y,z,u,v,w),dp;
ideal i=w,x,z,y,v;
sortier(i);
↦ _[1]=w
↦ _[2]=v
↦ _[3]=z
↦ _[4]=y
↦ _[5]=x
```

D.7.3 rinvar_lib

Library: rinvar.lib

Purpose: Invariant Rings of Reductive Groups

Author: Thomas Bayer, tbayer@in.tum.de
http://wwwmayr.informatik.tu-muenchen.de/personen/bayert/ Current Address: Institut fuer Informatik, TU Muenchen

Overview: Implementation based on Derksen's algorithm. Written in the scope of the diploma thesis (advisor: Prof. Gert-Martin Greuel) 'Computations of moduli spaces of semiquasihomogenous singularities and an implementation in Singular'

Procedures: See also: Section D.6.18 [qhmoduli_lib], page 1741; Section D.8.10 [zeroset_lib], page 1898.

D.7.3.1 HilbertSeries

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: HilbertSeries(I, w); ideal I, intvec wt

Purpose: compute the polynomial p of the Hilbert Series, represented by p/q, of the ring K[t_1,...,t_m,y_1,...,y_r]/I1 where 'w' are the weights of the variables, computed, e.g., by 'HilbertWeights', 'I1' is of the form I[1] - y_1,...,I[r] - y_r and is quasihomogenous w.r.t. 'w'

Return: intvec

Note: the leading 0 of the result does not belong to p, but is needed in the Hilbert driven 'std'.

D.7.3.2 HilbertWeights

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Purpose: compute the weights of the "slack" variables needed for the computation of the algebraic relations of the generators of 'I' s.t. the Hilbert driven 'std' can be used.

Return: intvec

Assume: basering = K[t_1,...,t_m,...], 'I' is quasihomogenous w.r.t. 'w' and contains only polynomials in t_1,...,t_m

D.7.3.3 ImageVariety

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: ImageVariety(ideal I, F [, w]);ideal I; F is a list/ideal, intvec w.

Purpose: compute the Zariski closure of the image of the variety of I under the morphism F.

Note: if 'I' and 'F' are quasihomogenous w.r.t. 'w' then the Hilbert-driven 'std' is used.

Return: polynomial ring over the same ground field, containing the ideal 'imageid'. The variables are Y(1),...,Y(k) where k = size(F) - 'imageid' is the ideal of the Zariski closure of F(X) where X is the variety of I.

Example:

```
LIB "rinvar.lib";
ring B   = 0,(x,y),dp;
ideal I  = x4 - y4;
ideal F  = x2, y2, x*y;
def R = ImageVariety(I, F);
↦
↦ // 'ImageVariety' created a new ring.
↦ // To see the ring, type (if the name 'R' was assigned to the return valu\
   e):
↦      show(R);
↦ // To access the ideal of the image variety, type
↦      setring R;  imageid;
↦
setring R;
imageid;
↦ imageid[1]=Y(1)*Y(2)-Y(3)^2
↦ imageid[2]=Y(1)^2-Y(2)^2
↦ imageid[3]=Y(2)^3-Y(1)*Y(3)^2
```

D.7.3.4 ImageGroup

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: ImageGroup(G, action); ideal G, action;

Purpose: compute the ideal of the image of G in GL(m,K) induced by the linear action 'action', where G is an algebraic group and 'action' defines an action of G on K^m (size(action) = m).

Return: ring, a polynomial ring over the same ground field as the basering, containing the ideals 'groupid' and 'actionid'.
- 'groupid' is the ideal of the image of G (order <= order of G) - 'actionid' defines the linear action of 'groupid' on K^m.

Note: 'action' and 'actionid' have the same orbits
all variables which give only rise to 0's in the m x m matrices of G have been omitted.

Assume: basering K[s(1..r),t(1..m)] has r + m variables, G is the ideal of an algebraic group and F is an action of G on K^m. G contains only the variables s(1)...s(r). The action 'action' is given by polynomials f_1,...,f_m in basering, s.t. on the ring level we have
K[t_1,...,t_m] –> K[s_1,...,s_r,t_1,...,t_m]/G
t_i –> f_i(s_1,...,s_r,t_1,...,t_m)

Example:

```
LIB "rinvar.lib";
ring B   = 0,(s(1..2), t(1..2)),dp;
ideal G = s(1)^3-1, s(2)^10-1;
ideal action = s(1)*s(2)^8*t(1), s(1)*s(2)^7*t(2);
def R = ImageGroup(G, action);
↦
↦ // 'ImageGroup' created a new ring.
↦ // To see the ring, type (if the name 'R' was assigned to the return valu\
   e):
↦      show(R);
↦ // To access the ideal of the image of the input group and to access the \
   new
↦ // action of the group, type
↦      setring R;  groupid; actionid;
↦
setring R;
groupid;
↦ groupid[1]=-s(1)+s(2)^4
↦ groupid[2]=s(1)^8-s(2)^2
↦ groupid[3]=s(1)^7*s(2)^2-1
actionid;
↦ actionid[1]=s(1)*t(1)
↦ actionid[2]=s(2)*t(2)
```

D.7.3.5 InvariantRing

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: InvariantRing(G, Gact [, opt]); ideal G, Gact; int opt

Purpose: compute generators of the invariant ring of G w.r.t. the action 'Gact'

Assume: G is a finite group and 'Gact' is a linear action.

Return: ring R; this ring comes with the ideals 'invars' and 'groupid' and with the poly 'newA':
- 'invars' contains the algebra generators of the invariant ring - 'groupid' is the ideal of G in the new ring
- 'newA' is the new representation of the primitive root of the minimal polynomial of the ring which was active when calling the procedure (if the minpoly did not change, 'newA' is set to 'a').

Note: the minimal polynomial of the output ring depends on some random choices

Example:

```
LIB "rinvar.lib";
ring B = 0, (s(1..2), t(1..2)), dp;
ideal G = -s(1)+s(2)^3, s(1)^4-1;
ideal action = s(1)*t(1), s(2)*t(2);
def R = InvariantRing(std(G), action);
↦
↦ // 'InvariantRing' created a new ring.
↦ // To see the ring, type (if the name 'R' was assigned to the return valu\
   e):
↦      show(R);
↦ // To access the generators of the invariant ring type
↦      setring R; invars;
↦ // Note that the input group G is stored in R as the ideal 'groupid'; to
↦ // see it, type
↦     groupid;
↦ // Note that 'InvariantRing' might change the minimal polynomial
↦ // The representation of the algebraic number is given by 'newA'
↦
setring R;
invars;
↦ invars[1]=t(1)^4
↦ invars[2]=t(1)^3*t(2)^3
↦ invars[3]=t(1)^2*t(2)^6
↦ invars[4]=t(1)*t(2)^9
↦ invars[5]=t(2)^12
```

D.7.3.6 InvariantQ

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: InvariantQ(f, G, action); poly f; ideal G, action

Purpose: check whether the polynomial f is invariant w.r.t. G, where G acts via 'action' on K^m.

Assume: basering = K[s_1,...,s_m,t_1,...,t_m] where K = Q of K = Q(a) and minpoly != 0, f contains only t_1,...,t_m, G is the ideal of an algebraic group and a standardbasis.

Return: int;
0 if f is not invariant,
1 if f is invariant

Note: G need not be finite

D.7.3.7 LinearizeAction

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: LinearizeAction(G,action,r); ideal G, action; int r

Purpose: linearize the group action 'action' and find an equivariant embedding of K^m where m = size(action).

Assume: G contains only variables var(1..r) (r = nrs)
basering = K[s(1..r),t(1..m)], K = Q or K = Q(a) and minpoly != 0.

Return: polynomial ring containing the ideals 'actionid', 'embedid', 'groupid' - 'actionid' is the ideal defining the linearized action of G - 'embedid' is a parameterization of an equivariant embedding (closed) - 'groupid' is the ideal of G in the new ring

Note: set printlevel > 0 to see a trace

Example:

```
LIB "rinvar.lib";
ring B   = 0,(s(1..5), t(1..3)),dp;
ideal G =  s(3)-s(4), s(2)-s(5), s(4)*s(5), s(1)^2*s(4)+s(1)^2*s(5)-1, s(1)^2*s(5)^2
ideal action = -s(4)*t(1)+s(5)*t(1), -s(4)^2*t(2)+2*s(4)^2*t(3)^2+s(5)^2*t(2), s(4)*t
LinearActionQ(action, 5);
↦ 0
def R = LinearizeAction(G, action, 5);
↦ 
↦ // 'LinearizeAction' created a new ring.
↦ // To see the ring, type (if the name 'R' was assigned to the return valu\
   e):
↦      show(R);
↦ // To access the new action and the equivariant embedding, type
↦      setring R; actionid; embedid; groupid
↦ 
setring R;
R;
↦ // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    s(1) s(2) s(3) s(4) s(5) t(1) t(2) t(3) t(\
   4)
↦ //        block   2 : ordering C
actionid;
↦ actionid[1]=-s(4)*t(1)+s(5)*t(1)
↦ actionid[2]=-s(4)^2*t(2)+s(5)^2*t(2)+2*s(4)^2*t(4)
↦ actionid[3]=s(4)*t(3)+s(5)*t(3)
↦ actionid[4]=s(4)^2*t(4)+s(5)^2*t(4)
embedid;
↦ embedid[1]=t(1)
↦ embedid[2]=t(2)
↦ embedid[3]=t(3)
↦ embedid[4]=t(3)^2
groupid;
↦ groupid[1]=s(3)-s(4)
↦ groupid[2]=s(2)-s(5)
↦ groupid[3]=s(4)*s(5)
↦ groupid[4]=s(1)^2*s(4)+s(1)^2*s(5)-1
↦ groupid[5]=s(1)^2*s(5)^2-s(5)
↦ groupid[6]=s(4)^4-s(5)^4+s(1)^2
↦ groupid[7]=s(1)^4+s(4)^3-s(5)^3
↦ groupid[8]=s(5)^5-s(1)^2*s(5)
LinearActionQ(actionid, 5);
↦ 1
```

D.7.3.8 LinearActionQ

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: LinearActionQ(action,nrs); ideal action, int nrs

Purpose: check whether the action defined by 'action' is linear w.r.t. the variables var(nrs + 1...nvars(basering)).

Return: 0 action not linear
1 action is linear

Example:

```
LIB "rinvar.lib";
ring R   = 0,(s(1..5), t(1..3)),dp;
ideal G =  s(3)-s(4), s(2)-s(5), s(4)*s(5), s(1)^2*s(4)+s(1)^2*s(5)-1,
s(1)^2*s(5)^2-s(5), s(4)^4-s(5)^4+s(1)^2, s(1)^4+s(4)^3-s(5)^3,
s(5)^5-s(1)^2*s(5);
ideal Gaction = -s(4)*t(1)+s(5)*t(1),
-s(4)^2*t(2)+2*s(4)^2*t(3)^2+s(5)^2*t(2),
s(4)*t(3)+s(5)*t(3);
LinearActionQ(Gaction, 5);
↦ 0
LinearActionQ(Gaction, 8);
↦ 1
```

D.7.3.9 LinearCombinationQ

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: LinearCombination(I, f); ideal I, poly f

Purpose: test whether f can be written as a linear combination of the generators of I.

Return: 0 f is not a linear combination
1 f is a linear combination

D.7.3.10 MinimalDecomposition

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: MinimalDecomposition(f,a,b); poly f; int a, b.

Purpose: decompose f as a sum M[1,1]*M[2,1] + ... + M[1,r]*M[2,r] where M[1,i] contains only s(1..a), M[2,i] contains only t(1...b) s.t. r is minimal

Assume: f polynomial in K[s(1..a),t(1..b)], K = Q or K = Q(a) and minpoly != 0

Return: 2 x r matrix M s.t. f = M[1,1]*M[2,1] + ... + M[1,r]*M[2,r]

Example:

```
LIB "rinvar.lib";
ring R = 0, (s(1..2), t(1..2)), dp;
poly h = s(1)*(t(1) + t(1)^2) +  (t(2) + t(2)^2)*(s(1)^2 + s(2));
matrix M = MinimalDecomposition(h, 2, 2);
M;
↦ M[1,1]=s(1)^2+s(2)
↦ M[1,2]=s(1)
↦ M[2,1]=t(2)^2+t(2)
↦ M[2,2]=t(1)^2+t(1)
M[1,1]*M[2,1] + M[1,2]*M[2,2] - h;
↦ 0
```

D.7.3.11 NullCone

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: NullCone(G, action); ideal G, action

Purpose: compute the ideal of the nullcone of the linear action of G on K^n, given by 'action', by means of Deksen's algorithm

Assume: basering = K[s(1..r),t(1..n)], K = Q or K = Q(a) and minpoly != 0, G is an ideal of a reductive algebraic group in K[s(1..r)], 'action' is a linear group action of G on K^n (n = ncols(action))

Return: ideal of the nullcone of G.

Note: the generators of the nullcone are homogeneous, but in general not invariant

Example:

```
LIB "rinvar.lib";
ring R = 0, (s(1..2), x, y), dp;
ideal G = -s(1)+s(2)^3, s(1)^4-1;
ideal action = s(1)*x, s(2)*y;
ideal inv = NullCone(G, action);
inv;
↦ inv[1]=x^4
↦ inv[2]=x^3*y^3
↦ inv[3]=x^2*y^6
↦ inv[4]=x*y^9
↦ inv[5]=y^12
```

D.7.3.12 ReynoldsImage

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: ReynoldsImage(RO, f); list RO, poly f

Purpose: compute the Reynolds image of the polynomial f, where RO represents the Reynolds operator

Return: poly

D.7.3.13 ReynoldsOperator

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Usage: ReynoldsOperator(G, action [, opt]); ideal G, action; int opt

Purpose: compute the Reynolds operator of the group G which acts via 'action'

Return: polynomial ring R over a simple extension of the ground field of the basering (the extension might be trivial), containing a list 'ROelements', the ideals 'id', 'actionid' and the polynomial 'newA'. R = K(a)[s(1..r),t(1..n)].
- 'ROelements' is a list of ideals, each ideal represents a substitution map F : R -> R according to the zero-set of G - 'id' is the ideal of G in the new ring
- 'newA' is the new representation of a' in terms of a. If the basering does not contain a parameter then 'newA' = 'a'.

Assume: basering = K[s(1..r),t(1..n)], K = Q or K = Q(a') and minpoly != 0, G is the ideal of a finite group in K[s(1..r)], 'action' is a linear group action of G

D.7.3.14 SimplifyIdeal

Procedure from library `rinvar.lib` (see Section D.7.3 [rinvar_lib], page 1806).

Purpose: simplify ideal I to the ideal I', do not change the names of the first m variables, new ideal I' might contain less variables. I' contains variables var(1..m)

Return: list
_[1] ideal I'
_[2] ideal representing a map phi to a ring with probably less vars. s.th. phi(I) = I'
_[3] list of variables
_[4] list from 'elimpart'

D.7.4 invar_lib

Library: invar.lib

Purpose: Procedures to compute invariant rings of SL(n) and torus groups

Author: Harm Derksen, hderksen@math.unibas.ch

Procedures:

D.7.4.1 SL

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: SL(<int>)

Returns: SL(n) sets the current group to SL_n. The following global variables will be set:
group of type <ring>
groupideal of type <ideal>
SLrep of type <matrix>
reynolds of type <proc>
The quotient of of ‘group‘ and ‘groupideal‘ is the coordinate ring of SL_n. The matrix ‘SLrep‘ will be set to the standard representation of SL_n. The basering will be set to ’group’.

Example:

```
LIB "invar.lib";
SL(3);
Invar::group;
↦ // coefficients: QQ
↦ // number of vars : 9
↦ //        block   1 : ordering dp
↦ //                  : names    g(1) g(2) g(3) g(4) g(5) g(6) g(7) g(8) g(\
   9)
↦ //        block   2 : ordering C
groupideal;
↦ groupideal[1]=g(3)*g(5)*g(7)-g(2)*g(6)*g(7)-g(3)*g(4)*g(8)+g(1)*g(6)*g(8)\
   +g(2)*g(4)*g(9)-g(1)*g(5)*g(9)+1
print(SLrep);
↦ g(1),g(2),g(3),
↦ g(4),g(5),g(6),
↦ g(7),g(8),g(9)
```

D.7.4.2 torus

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: torus(<int>)

Returns: torus(n) sets the current group to an n-dimensional torus. The following global variables will be changed:
group of type <ring>
groupideal of type <ideal>
reynolds of type <proc>
The quotient of of ‘group‘ and ‘groupideal‘ is the coordinate ring of an n-dimensional torus. The basering will be set to ’group’.

Example:

```
LIB "invar.lib";
torus(3);
Invar::group;
↦ // coefficients: QQ
↦ // number of vars : 4
↦ //        block   1 : ordering dp
↦ //                  : names    g(1) g(2) g(3) g(4)
↦ //        block   2 : ordering C
groupideal;
↦ groupideal[1]=g(1)*g(2)*g(3)*g(4)-1
```

D.7.4.3 torusrep

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: torusrep(<list>), <list> must be a list of integer vectors of length n, where n is the dimension of the current torusgroup.

Returns: torusrep(m) gives a matrix with entries in 'group'. This matrix represents the action of the torus with weights
m[1],m[2],...,m[size(m)]

Example:

```
LIB "invar.lib";
torus(1);                    // Take the 1-dimensional torus, the multiplicative group
list weights=-2,-3,7;        // 3-dimensial action with weights -2,-3,7
matrix m=torusrep(weights);// compute the matrix of the representation
invar(m);                    // compute the invariant ring
↦
↦ Ideal B:
↦ x(1)^2*x(2)*x(3)-y(1)^2*y(2)*y(3),
↦ x(2)^2*y(1)^3-x(1)^3*y(2)^2,
↦ x(2)^3*x(3)*y(1)-x(1)*y(2)^3*y(3),
↦ x(1)^5*x(3)*y(2)-x(2)*y(1)^5*y(3),
↦ x(1)*x(2)^4*x(3)^2-y(1)*y(2)^4*y(3)^2,
↦ x(1)^7*x(3)^2-y(1)^7*y(3)^2,
↦ x(2)^7*x(3)^3-y(2)^7*y(3)^3
↦
↦ Zero Fiber Ideal:
↦ x(1)^2*x(2)*x(3),
↦ x(1)*x(2)^4*x(3)^2,
↦ x(1)^7*x(3)^2,
↦ x(2)^7*x(3)^3
↦
↦ Generating Invariants:
↦ x(1)^2*x(2)*x(3),
↦ x(1)*x(2)^4*x(3)^2,
↦ x(1)^7*x(3)^2,
↦ x(2)^7*x(3)^3
```

D.7.4.4 finiterep

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: finiterep(<list>), <list> must be a list of matrices

Returns: finiterep(m) gives a matrix with coefficients in the ring 'group' which represents the action of the finite group where the elements of the finite group act as m[1],m[2],...m[size(m)].

Example:

```
LIB "invar.lib";
finite(6);                              // The symmetric group S_3
matrix id=unitmat(3);                   // identity matrix
matrix m3[3][3]=0,1,0,0,0,1,1,0,0;      // corresponds with (1 2 3)
matrix m2[3][3]=0,1,0,1,0,0,0,0,1;      // corresponds with (1 2)
list a=id,m3,m3*m3,m2,m2*m3,m2*m3*m3;   // all elements of S_3
matrix rep=finiterep(a);                // compute matrix of standard repr.
invar(rep);                             // compute the invariant ring
↦
↦ Ideal B:
↦ x(1)+x(2)+x(3)-y(1)-y(2)-y(3),
↦ 36*x(2)^2-115*x(1)*x(3)-79*x(2)*x(3)-79*x(3)^2+24*x(1)*y(1)-12*x(2)*y(1)+\
  103*x(3)*y(1)-24*y(1)^2-36*x(2)*y(2)+79*x(3)*y(2)+12*y(1)*y(2)+310*x(1)*y\
  (3)+274*x(2)*y(3)+389*x(3)*y(3)-298*y(1)*y(3)-274*y(2)*y(3)-310*y(3)^2,
↦ 1296*x(3)^3+1584*x(1)*x(3)*y(1)+1584*x(2)*x(3)*y(1)+288*x(3)^2*y(1)-1584*\
  x(3)*y(1)^2+2112*x(1)^2*y(2)+4224*x(1)*x(2)*y(2)+2112*x(2)^2*y(2)+6648*x(\
  1)*x(3)*y(2)+6648*x(2)*x(3)*y(2)+3240*x(3)^2*y(2)-2112*x(1)*y(1)*y(2)-211\
  2*x(2)*y(1)*y(2)-4824*x(3)*y(1)*y(2)-2112*x(1)*y(2)^2-2112*x(2)*y(2)^2-45\
  36*x(3)*y(2)^2+5248*x(1)^2*y(3)+13645*x(1)*x(2)*y(3)+8397*x(2)^2*y(3)+448\
  95*x(1)*x(3)*y(3)+48044*x(2)*x(3)*y(3)+38351*x(3)^2*y(3)-6832*x(1)*y(1)*y\
  (3)-9981*x(2)*y(1)*y(3)-41519*x(3)*y(1)*y(3)+1584*y(1)^2*y(3)-16120*x(1)*\
  y(2)*y(3)-19269*x(2)*y(2)*y(3)-51647*x(3)*y(2)*y(3)+9048*y(1)*y(2)*y(3)+8\
  760*y(2)^2*y(3)-26722*x(1)*y(3)^2-29871*x(2)*y(3)^2-61121*x(3)*y(3)^2+230\
  58*y(1)*y(3)^2+30234*y(2)*y(3)^2+21474*y(3)^3
↦
↦ Zero Fiber Ideal:
↦ x(1)+x(2)+x(3),
↦ 36*x(2)^2-115*x(1)*x(3)-79*x(2)*x(3)-79*x(3)^2,
↦ x(3)^3
↦
↦ Generating Invariants:
↦ x(1)+x(2)+x(3),
↦ -43/3*x(1)^2-194/3*x(1)*x(2)-43/3*x(2)^2-194/3*x(1)*x(3)-194/3*x(2)*x(3)-\
  43/3*x(3)^2,
↦ 1/3*x(1)^3+1/3*x(2)^3+1/3*x(3)^3
```

D.7.4.5 sympower

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: sympower(<matrix>,<int>)

Returns: If m is a matrix with coefficients in the ring 'group', representing the action on some vector space V, then sympower(m,n) gives the matrix of the representation of the group on the n-th symmetric power of V.

Example:

```
LIB "invar.lib";
SL(2);
print(SLrep);
↦ g(1),g(2),
↦ g(3),g(4)
print(sympower(SLrep,3));
↦ g(1)^3,    3*g(1)^2*g(2),      3*g(1)*g(2)^2,      g(2)^3,
↦ g(1)^2*g(3),3*g(1)^2*g(4)-2*g(1), 3*g(1)*g(2)*g(4)-g(2),g(2)^2*g(4),
↦ g(1)*g(3)^2,3*g(1)*g(3)*g(4)-g(3),3*g(1)*g(4)^2-2*g(4), g(2)*g(4)^2,
↦ g(3)^3,     3*g(3)^2*g(4),        3*g(3)*g(4)^2,        g(4)^3
```

D.7.4.6 invar

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: invar(<matrix>)

Returns: If m is a n x n matrix with coefficients in the ring 'group', representing the action on some vector space V, then invar(m); gives polynomials in x(1),x(2),...,x(n) who generate the invariant ring. The following global variables will be set:
polyring of type <ring> polynomial ring in x(1),...,x(n) invring of type <ideal> entries generate the inv. ring representation of type <matrix>
The base ring will be set to 'polyring' which is a global variable representing the polynomial ring on which the group acts. The variable 'representation' will be set to the input m.

Example:

```
LIB "invar.lib";
SL(2);                          // Take the group SL_2
matrix m=dsum(SLrep,SLrep,SLrep,SLrep);
// 4 copies of the standard representation
invar(m);                       // empirical evidence for FFT
↦ 
↦ Ideal B:
↦ x(6)*x(7)-x(5)*x(8)-y(6)*y(7)+y(5)*y(8),
↦ x(4)*x(7)-x(3)*x(8)-y(4)*y(7)+y(3)*y(8),
↦ x(2)*x(7)-x(1)*x(8)-y(2)*y(7)+y(1)*y(8),
↦ x(4)*x(5)-x(3)*x(6)-y(4)*y(5)+y(3)*y(6),
↦ x(2)*x(5)-x(1)*x(6)-y(2)*y(5)+y(1)*y(6),
↦ x(2)*x(3)-x(1)*x(4)-y(2)*y(3)+y(1)*y(4),
↦ x(8)*y(4)*y(5)-x(8)*y(3)*y(6)-x(6)*y(4)*y(7)+x(4)*y(6)*y(7)+x(6)*y(3)*y(8\
   )-x(4)*y(5)*y(8),
↦ x(7)*y(4)*y(5)-x(7)*y(3)*y(6)-x(5)*y(4)*y(7)+x(3)*y(6)*y(7)+x(5)*y(3)*y(8\
   )-x(3)*y(5)*y(8),
↦ x(8)*y(2)*y(5)-x(8)*y(1)*y(6)-x(6)*y(2)*y(7)+x(2)*y(6)*y(7)+x(6)*y(1)*y(8\
   )-x(2)*y(5)*y(8),
↦ x(7)*y(2)*y(5)-x(7)*y(1)*y(6)-x(5)*y(2)*y(7)+x(1)*y(6)*y(7)+x(5)*y(1)*y(8\
   )-x(1)*y(5)*y(8),
↦ x(8)*y(2)*y(3)-x(8)*y(1)*y(4)-x(4)*y(2)*y(7)+x(2)*y(4)*y(7)+x(4)*y(1)*y(8\
   )-x(2)*y(3)*y(8),
↦ x(7)*y(2)*y(3)-x(7)*y(1)*y(4)-x(3)*y(2)*y(7)+x(1)*y(4)*y(7)+x(3)*y(1)*y(8\
   )-x(1)*y(3)*y(8),
↦ x(6)*y(2)*y(3)-x(6)*y(1)*y(4)-x(4)*y(2)*y(5)+x(2)*y(4)*y(5)+x(4)*y(1)*y(6\
   )-x(2)*y(3)*y(6),
↦ x(5)*y(2)*y(3)-x(5)*y(1)*y(4)-x(3)*y(2)*y(5)+x(1)*y(4)*y(5)+x(3)*y(1)*y(6\
   )-x(1)*y(3)*y(6)
↦ 
↦ Zero Fiber Ideal:
↦ x(6)*x(7)-x(5)*x(8),
↦ x(4)*x(7)-x(3)*x(8),
↦ x(2)*x(7)-x(1)*x(8),
↦ x(4)*x(5)-x(3)*x(6),
↦ x(2)*x(5)-x(1)*x(6),
↦ x(2)*x(3)-x(1)*x(4)
↦
↦ Generating Invariants:
↦ x(6)*x(7)-x(5)*x(8),
↦ x(4)*x(7)-x(3)*x(8),
↦ x(2)*x(7)-x(1)*x(8),
↦ x(4)*x(5)-x(3)*x(6),
↦ x(2)*x(5)-x(1)*x(6),
↦ x(2)*x(3)-x(1)*x(4)
setring Invar::polyring;
Invar::reynolds(x(1)*x(4));            // The reynolds operator is computed using
↦ -1/2*x(2)*x(3)+1/2*x(1)*x(4)
// the Omega process.
```

D.7.4.7 SLreynolds

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

Usage: SLreynolds(f) poly f

Returns: the reynolds operator applied to f

Note: if the group is SL_n the omega-process is used

D.7.4.8 torusreynolds

Procedure from library `invar.lib` (see Section D.7.4 [invar_lib], page 1812).

D.7.5 stratify_lib

Library: stratify.lib

Purpose: Algorithmic Stratification for Unipotent Group-Actions

Author: Anne Fruehbis-Krueger, anne@mathematik.uni-kl.de

Procedures:

D.7.5.1 prepMat

Procedure from library `stratify.lib` (see Section D.7.5 [stratify_lib], page 1817).

Usage: prepMat(M,wr,ws,step);
where M is a matrix, wr is an intvec of size ncols(M), ws an intvec of size nrows(M) and step is an integer

Return: 2 lists of submatrices corresponding to the filtrations specified by wr and ws:
the first list corresponds to the list for the filtration of AdA, i.e. the ranks of these matrices will be the r_i, the second one to the list for the filtration of L, i.e. the ranks of these matrices will be the s_i

Note: * the entries of the matrix M are M_ij=delta_i(x_j),
* wr is used to determine what subset of the set of all dx_i is generating AdF^l(A):
if (k-1)*step <= wr[i] < k*step, then dx_i is in the set of generators of AdF^l(A) for all l>=k and the i-th column of M appears in each submatrix starting from the k-th

* ws is used to determine what subset of the set of all delta_i is generating Z_l(L): if (k-1)*step <= ws[i] < k*step, then delta_i is in the set of generators of Z_l(A) for l < k and the i-th row of M appears in each submatrix up to the (k-1)th
* the entries of wr and ws as well as step should be positive integers

Example:

```
LIB "stratify.lib";
ring r=0,(t(1..3)),dp;
matrix M[2][3]=0,t(1),3*t(2),0,0,t(1);
print(M);
↦ 0,t(1),3*t(2),
↦ 0,0,   t(1)
intvec wr=1,3,5;
intvec ws=2,4;
int step=2;
prepMat(M,wr,ws,step);
↦ [1]:
↦    [1]:
↦       _[1,1]=0
↦       _[2,1]=0
↦    [2]:
↦       _[1,1]=0
↦       _[1,2]=t(1)
↦       _[2,1]=0
↦       _[2,2]=0
↦    [3]:
↦       _[1,1]=0
↦       _[1,2]=t(1)
↦       _[1,3]=3*t(2)
↦       _[2,1]=0
↦       _[2,2]=0
↦       _[2,3]=t(1)
↦ [2]:
↦    [1]:
↦       _[1,1]=0
↦       _[1,2]=t(1)
↦       _[1,3]=3*t(2)
↦       _[2,1]=0
↦       _[2,2]=0
↦       _[2,3]=t(1)
↦    [2]:
↦       _[1,1]=0
↦       _[1,2]=0
↦       _[1,3]=t(1)
```

D.7.5.2 stratify

Procedure from library `stratify.lib` (see Section D.7.5 [stratify_lib], page 1817).

Usage: stratify(M,wr,ws,step);
where M is a matrix, wr is an intvec of size ncols(M), ws an intvec of size nrows(M) and step is an integer

Return: list of lists, each entry of the big list corresponds to one locally closed set and has the following entries:

1) intvec giving the corresponding rs-vector
2) ideal determining the closed set
3) list d of polynomials determining the open set D(d[1]) empty list if there is more than one open set
4-n) lists of polynomials determining open sets which all lead to the same rs-vector

Note: * ring ordering should be global, i.e. the ring should be a polynomial ring
* the entries of the matrix M are M_ij=delta_i(x_j),
* wr is used to determine what subset of the set of all dx_i is generating AdF^l(A):
if (k-1)*step < wr[i] <= k*step, then dx_i is in the set of generators of AdF^l(A) for all l>=k
* ws is used to determine what subset of the set of all delta_i is generating Z_l(L):
if (k-1)*step <= ws[i] < k*step, then delta_i is in the set of generators of Z_l(A) for l < k
* the entries of wr and ws as well as step should be positive integers
* the filtrations have to be known, no sanity checks concerning the filtrations are performed !!!

Example:

```
LIB "stratify.lib";
ring r=0,(t(1..3)),dp;
matrix M[2][3]=0,t(1),3*t(2),0,0,t(1);
intvec wr=1,3,5;
intvec ws=2,4;
int step=2;
stratify(M,wr,ws,step);
↦ [1]:
↦    [1]:
↦       0,0,0,0
↦    [2]:
↦       _[1]=t(2)
↦       _[2]=t(1)
↦    [3]:
↦       [1]:
↦          1
↦ [2]:
↦    [1]:
↦       0,1,0,1
↦    [2]:
↦       _[1]=t(1)
↦    [3]:
↦       [1]:
↦          t(2)
↦       [2]:
↦          t(2)
↦ [3]:
↦    [1]:
↦       1,2,1,2
↦    [2]:
↦       _[1]=0
↦    [3]:
↦       [1]:
↦          t(1)
↦         [2]:
↦            t(1)
```

D.8 Symbolic-numerical solving

D.8.1 ffsolve_lib

Library: ffsolve.lib

Purpose: multivariate equation solving over finite fields

Author: Gergo Gyula Borus, borisz@borisz.net

Procedures:

D.8.1.1 ffsolve

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: ffsolve(I[, L]); I ideal, L list of strings

Return: list L, the common roots of I as ideal

Assume: basering is a finite field of type (p^n,a)

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
ideal I;
I[1]=x(1)^2*x(2)+(a)*x(1)*x(2)^2+(a+1);
I[2]=x(1)^2*x(2)*x(3)^2+(a)*x(1);
I[3]=(a+1)*x(1)*x(3)+(a+1)*x(1);
ffsolve(I);
↦ [1]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a)
↦    _[3]=x(1)+1
↦ [2]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a+1)
↦    _[3]=x(1)+(a+1)
```

D.8.1.2 PEsolve

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: PEsolve(I[, i]); I ideal, i optional integer
solve I (system of multivariate equations) over a
finite field using an equvalence property when i is
not given or set to 2, otherwise if i is set to 0
then check whether common roots exists

Return: list if optional parameter is not given or set to 2,
integer if optional is set to 0

Assume: basering is a finite field of type (p^n,a)

Note: When the optional parameter is set to 0, speoff only checks if I has common roots, then return 1, otherwise return 0.

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
ideal I;
I[1]=x(1)^2*x(2)+(a)*x(1)*x(2)^2+(a+1);
I[2]=x(1)^2*x(2)*x(3)^2+(a)*x(1);
I[3]=(a+1)*x(1)*x(3)+(a+1)*x(1);
PEsolve(I);
↦ // ** redefining I (  ideal I = defaultIdeal();) ffsolve.lib::productOfEq\
   s:1001
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ // ** redefining res (          ideal res;) ffsolve.lib::PEsolve:273
↦ // ** redefining new (      list new = increment(start,i);) ffsolve.lib::\
   melyseg:969
↦ // ** redefining l (int l=1;) ffsolve.lib::melyseg:970
↦ [1]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a)
↦    _[3]=x(1)+1
↦ [2]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a+1)
↦    _[3]=x(1)+(a+1)
```

D.8.1.3 simplesolver

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: simplesolver(I); I ideal
solve I (system of multivariate equations) over a
finite field by exhausting search

Return: list L, the common roots of I as ideal

Assume: basering is a finite field of type (p^n,a)

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
ideal I;
I[1]=x(1)^2*x(2)+(a)*x(1)*x(2)^2+(a+1);
I[2]=x(1)^2*x(2)*x(3)^2+(a)*x(1);
I[3]=(a+1)*x(1)*x(3)+(a+1)*x(1);
simplesolver(I);
↦ [1]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a)
↦    _[3]=x(1)+1
↦ [2]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a+1)
↦    _[3]=x(1)+(a+1)
```

D.8.1.4 GBsolve

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: GBsolve(I); I ideal
solve I (system of multivariate equations) over an
extension of Z/p by Groebner basis methods

Return: list L, the common roots of I as ideal

Assume: basering is a finite field of type (p^n,a)

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
ideal I;
I[1]=x(1)^2*x(2)+(a)*x(1)*x(2)^2+(a+1);
I[2]=x(1)^2*x(2)*x(3)^2+(a)*x(1);
I[3]=(a+1)*x(1)*x(3)+(a+1)*x(1);
```

```
GBsolve(I);
↦ [1]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a)
↦    _[3]=x(1)+1
↦ [2]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a+1)
↦    _[3]=x(1)+(a+1)
```

D.8.1.5 XLsolve

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: XLsolve(I[, d]); I ideal, d optional integer
solve I (system of multivariate polynomials) with a
variant of the linearization technique, multiplying
the polynomials with monomials of degree at most d
(default is 2)

Return: list L of the common roots of I as ideals

Assume: basering is a finite field of type (p^n,a)

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
ideal I;
I[1]=(a)*x(1)^2+x(2)^2+(a+1);
I[2]=(a)*x(1)^2+(a)*x(1)*x(3)+(a)*x(2)^2+1;
I[3]=(a)*x(1)*x(3)+1;
I[4]=x(1)^2+x(1)*x(3)+(a);
XLsolve(I, 3);
↦ [1]:
↦    _[1]=x(3)+(a+1)
↦    _[2]=x(2)+1
↦    _[3]=x(1)+1
```

D.8.1.6 ZZsolve

Procedure from library `ffsolve.lib` (see Section D.8.1 [ffsolve_lib], page 1820).

Usage: ZZsolve(I); I ideal
solve I (system of multivariate equations) over a
finite field by mapping the polynomials to a single
univariate polynomial over extension of the basering

Return: list, the common roots of I as ideal

Assume: basering is a finite field of type (p^n,a)

Example:

```
LIB "ffsolve.lib";
ring R = (2,a),x(1..3),lp;
minpoly=a2+a+1;
```

```
ideal I;
I[1]=x(1)^2*x(2)+(a)*x(1)*x(2)^2+(a+1);
I[2]=x(1)^2*x(2)*x(3)^2+(a)*x(1);
I[3]=(a+1)*x(1)*x(3)+(a+1)*x(1);
ZZsolve(I);
↦ [1]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a)
↦    _[3]=x(1)+1
↦ [2]:
↦    _[1]=x(3)+1
↦    _[2]=x(2)+(a+1)
↦    _[3]=x(1)+(a+1)
```

D.8.2 interval_lib

Library: interval.lib

Purpose: implements interval arithmetic on polynomials

Authors: Dominik Bendle
Clara Petroll

Overloads:

// intervals
[intervalGet indexing
= intervalAssign assigning
== intervalEqual equality
print intervalPrint pretty print
+ intervalAdd addition
- intervalNegate negation (unary)
- intervalSubtract subtraction
* intervalMultiply multiplication
/ intervalDivide division
^ intervalPotentiate potentiation

// boxes
= boxSet assigning
[boxGet indexing
== boxEqual equality
print boxPrint printing
- boxSubtract subraction
intersect boxIntersect intersection

// intervalmatrices
[ivmatGet indexing
print ivmatPrint printing
nrows ivmatNrows number of rows
ncols ivmatNcols number of columns
det determinant determinant
* ivmatMultiply matrix multiplication

Procedures: See also: Section D.8.8 [rootisolation_lib], page 1889.

D.8.2.1 length2

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: length2(I), I interval

Return: length/size in measure sense

Example:

```
LIB "interval.lib";
ring R = 0,x,lp;
interval I = bounds2(0, 1);    I;
↦ [0, 1]
↦
length2(I);
↦ 1
I = bounds2(-1/2, 3/7);        I;
↦ [-1/2, 3/7]
↦
length2(I);
↦ 13/14
```

D.8.2.2 bounds2

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: bounds2(a, b), a, b number

Return: interval [a, b].

D.8.2.3 intervalmatrixInit

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: intervalmatrixInit(m, n) m, n int

Return: mxn matrix of [0,0]-intervals

Example:

```
LIB "interval.lib";
ring R = 0,x(1..5),lp;
ivmat A = intervalmatrixInit(4, 5); A;
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦
```

D.8.2.4 unitMatrix2

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: unitMatrix2(n)

Return: nxn unit matrix

Example:

```
LIB "interval.lib";
ring R = 0,x,lp;
ivmat E = unitMatrix2(4); E;
↦ [1, 1],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[1, 1],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[1, 1],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[1, 1]
↦
```

D.8.2.5 applyMatrix

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: A * b, A ivmat, b box

Return: A*b

Example:

```
LIB "interval.lib";
ring R = 0,(x,y,z),lp;
ideal I = xyz3+z2y2+x,x4+y3+2z+3,xyz+1/2;
interval J = bounds2(1/2, 3/2);
box B = list(J,J,J);
ivmat A = evalJacobianAtBox(I, B); A;
↦ [17/16, 97/16],[5/16, 189/16],[7/16, 351/16]
↦ [1/2, 27/2],[3/4, 27/4],[2, 2]
↦ [1/4, 9/4],[1/4, 9/4],[1/4, 9/4]
↦
A*B;
↦ [29/32, 1911/32],[13/8, 267/8],[3/8, 81/8]
↦
unitMatrix2(3) * B;
↦ [1/2, 3/2],[1/2, 3/2],[1/2, 3/2]
↦
diagMatrix(3, bounds2(0, 1)) * B;
↦ [0, 3/2],[0, 3/2],[0, 3/2]
↦
```

D.8.2.6 ivmatGaussian2

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: ivmatGaussian2(A) A ivmat

Return: 0 if A not invertible, 1,Ainv if A invertible

Example:

```
LIB "interval.lib";
ring R = 0,(x,y),lp;
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;
box B = list(bounds2(7/8, 9/8), bounds2(-1/10, 1/20));
ivmat J = evalJacobianAtBox (I, B); J;
↦ [69/20, 23/5],[-61/40, -27/40]
↦ [67/20, 24/5],[-159/40, -93/40]
↦
list result = ivmatGaussian2(J);
```

```
ivmat Jinv = result[2];
Jinv;
↦ [1060/4273, 620/187],[-1220/561, -180/4273]
↦ [2680/12819, 1280/187],[-920/187, -3680/12819]
↦
Jinv * J;
↦ [-7657037/799051, 12073835/799051],[-3961862/799051, 6773506/799051]
↦ [-18293234/799051, 73172936/2397153],[-7807572/799051, 15513600/799051]
↦
ivmat Jadj = diagMatrix(2, 1/det(J)) * adjunct(J);
Jadj;
↦ [620/4273, 1060/187],[-1220/561, -180/4273]
↦ [2680/12819, 1280/187],[-3680/561, -920/4273]
↦
Jadj * J;
↦ [-7940903/799051, 20722387/799051],[-6829045/799051, 6829045/799051]
↦ [-24583090/799051, 24583090/799051],[-7940903/799051, 20722387/799051]
↦
```

D.8.2.7 evalPolyAtBox2

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: evalPolyAtBox2(f, B), f poly, B box

Return: interval extension ff(intervals)

Example:

```
LIB "interval.lib";
ring R = 0,x,lp;
interval I1 = bounds2(0, 1); I1;
↦ [0, 1]
↦
poly f = x3 + 4x + 3;
evalPolyAtBox2(f, list(I1));
↦ [3, 8]
↦
ring S = 0,(x,y,z),lp;
interval I2 = bounds2(0, 1);
box B = list(I2, I2, I2);
poly f = xyz2 + 2x2 + (3/2)*y3x + z + 1;
evalPolyAtBox2(f, B);
↦ [1, 13/2]
↦
```

D.8.2.8 exclusionTest

Procedure from library `interval.lib` (see Section D.8.2 [interval_lib], page 1824).

Usage: exclusion test for roots with interval arithmetic

Return: list of boxes

Example:

```
LIB "interval.lib";
ring R = 0,(x,y),lp;
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;  // V(I) has four elements
interval i = bounds2(-3/2,3/2);
box B = list(i, i);
list result = exclusionTest(I, B, 1/512);
size(result[1]);
↦ 0
size(result[2]);
↦ 4
```

D.8.3 presolve_lib

Library: presolve.lib

Purpose: Pre-Solving of Polynomial Equations

Author: Gert-Martin Greuel, email: greuel@mathematik.uni-kl.de,

Procedures:

D.8.3.1 degreepart

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: degreepart(id,d1,d2[,v]); id=ideal/module, d1,d1=integers, v=intvec

Return: list of size 2,
_[1]: generators of id of [v-weighted] total degree >= d1 and <= d2 (default: v = 1,...,1)
_[2]: remaining generators of id

Note: if id is of type int/number/poly it is converted to ideal, if id is of type intmat/matrix/vector to module and then the corresponding generators are computed

Example:

```
LIB "presolve.lib";
ring r=0,(x,y,z),dp;
ideal i=1+x+x2+x3+x4,3,xz+y3+z8;
degreepart(i,0,4);
↦ [1]:
↦    _[1]=x4+x3+x2+x+1
↦    _[2]=3
↦ [2]:
↦    _[1]=z8+y3+xz
module m=[x,y,z],x*[x3,y2,z],[1,x2,z3,0,1];
intvec v=2,3,6;
show(degreepart(m,8,8,v));
↦ // list, 2 element(s):
↦ [1]:
↦    // module, 1 generator(s)
↦ [x4,xy2,xz]
↦ [2]:
↦    // module, 2 generator(s)
↦ [x,y,z]
↦ [1,x2,z3,0,1]
```

D.8.3.2 elimlinearpart

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: elimlinearpart(i[,n]); i=ideal, n=integer,
default: n=nvars(basering)

Return: list L with 5 entries:
L[1]: ideal obtained from i by substituting from the first n variables those which appear in a linear part of i, by putting this part into triangular form
L[2]: ideal of variables which have been substituted
L[3]: ideal, j-th element defines substitution of j-th var in [2]
L[4]: ideal of variables of basering, eliminated ones are set to 0
L[5]: ideal, describing the map from the basering to itself such that L[1] is the image of i

Note: the procedure always interreduces the ideal i internally w.r.t. ordering dp.

Example:

```
LIB "presolve.lib";
ring s=0,(u,x,y,z),dp;
ideal i = u3+y3+z-x,x2y2+z3,y+z+1,y+u;
elimlinearpart(i);
↦ [1]:
↦    _[1]=z4+3z3+z2
↦ [2]:
↦    _[1]=u
↦    _[2]=x
↦    _[3]=y
↦ [3]:
↦    _[1]=u-z-1
↦    _[2]=x-z
↦    _[3]=y+z+1
↦ [4]:
↦    _[1]=0
↦    _[2]=0
↦    _[3]=0
↦    _[4]=z
↦ [5]:
↦    _[1]=z+1
↦    _[2]=z
↦    _[3]=-z-1
↦    _[4]=z
```

D.8.3.3 elimpart

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: elimpart(i [,n,e]); i=ideal, n,e=integers
n : only the first n vars are considered for substitution,
e =0: substitute from linear part of i (same as elimlinearpart)
e!=0: eliminate also by direct substitution
(default: n = nvars(basering), e = 1)

Return: list of 5 objects:

[1]: ideal obtained by substituting from the first n variables those
from i, which appear in the linear part of i (or, if e!=0, which
can be expressed directly in the remaining vars)
[2]: ideal, variables which have been substituted
[3]: ideal, i-th element defines substitution of i-th var in [2]
[4]: ideal of variables of basering, substituted ones are set to 0
[5]: ideal, describing the map from the basering, say k[x(1..m)], to
itself onto k[..variables from [4]..] and [1] is the image of i

The ideal i is generated by [1] and [3] in k[x(1..m)], the map [5] maps [3] to 0, hence induces an isomorphism

k[x(1..m)]/i -> k[..variables from [4]..]/[1]

Note: Applying elimpart to interred(i) may result in more substitutions. However, interred may be more expansive than elimpart for big ideals

Example:

```
LIB "presolve.lib";
ring s=0,(u,x,y,z),dp;
ideal i = xy2-xu4-x+y2,x2y2+z3+zy,y+z2+1,y+u2;
elimpart(i);
↦ [1]:
↦    _[1]=u2-z2-1
↦    _[2]=u12-u2z+z3
↦ [2]:
↦    _[1]=y
↦    _[2]=x
↦ [3]:
↦    _[1]=u2+y
↦    _[2]=-u4+x
↦ [4]:
↦    _[1]=u
↦    _[2]=0
↦    _[3]=0
↦    _[4]=z
↦ [5]:
↦    _[1]=u
↦    _[2]=u4
↦    _[3]=-u2
↦    _[4]=z
i = interred(i); i;
↦ i[1]=z2+y+1
↦ i[2]=y2-x
↦ i[3]=u2+y
↦ i[4]=x3+z3+yz
elimpart(i);
↦ [1]:
↦    _[1]=u2-z2-1
↦    _[2]=u12-u2z+z3
↦ [2]:
↦    _[1]=x
↦    _[2]=y
↦ [3]:
↦    _[1]=-y2+x
↦    _[2]=u2+y
↦ [4]:
↦    _[1]=u
↦    _[2]=0
↦    _[3]=0
↦    _[4]=z
↦ [5]:
↦    _[1]=u
↦    _[2]=u4
↦    _[3]=-u2
↦    _[4]=z
elimpart(i,2);
↦ [1]:
↦    _[1]=z2+y+1
↦    _[2]=u2+y
↦    _[3]=y6+z3+yz
↦ [2]:
↦    _[1]=x
↦ [3]:
↦    _[1]=-y2+x
↦ [4]:
↦    _[1]=u
↦    _[2]=0
↦    _[3]=y
↦    _[4]=z
↦ [5]:
↦    _[1]=u
↦    _[2]=y2
↦    _[3]=y
↦    _[4]=z
```

D.8.3.4 elimpartanyr

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: elimpartanyr(i [,p,e]); i=ideal, p=polynomial, e=integer
p: product of vars to be eliminated,
e =0: substitute from linear part of i (same as elimlinearpart)
e!=0: eliminate also by direct substitution
(default: p=product of all vars, e=1)

Return: list of 6 objects:
[1]: (interreduced) ideal obtained by substituting from i those vars appearing in p, which occur in the linear part of i (or which can be expressed directly in the remaining variables, if e!=0)
[2]: ideal, variables which have been substituted
[3]: ideal, i-th element defines substitution of i-th var in [2]
[4]: ideal of variables of basering, substituted ones are set to 0
[5]: ideal, describing the map from the basering, say k[x(1..m)], to itself onto k[..variables fom [4]..] and [1] is the image of i
[6]: int, # of vars considered for substitution (= # of factors of p)

The ideal i is generated by [1] and [3] in k[x(1..m)], the map [5] maps [3] to 0, hence induces an isomorphism

k[x(1..m)]/i -> k[..variables fom [4]..]/[1]

Note: the procedure creates a ring with ordering dp and vars placed correctly and then applies `elimpart`.

Example:

```
LIB "presolve.lib";
ring s=0,(x,y,z),dp;
ideal i = x3+y2+z,x2y2+z3,y+z+1;
elimpartanyr(i,z);
↦ [1]:
↦    _[1]=x3+y2-y-1
↦    _[2]=x2y2-y3-3y2-3y-1
↦ [2]:
↦    _[1]=z
↦ [3]:
↦    _[1]=y+z+1
↦ [4]:
↦    _[1]=0
↦    _[2]=x
↦    _[3]=y
↦ [5]:
↦    _[1]=-y-1
↦    _[2]=x
↦    _[3]=y
↦ [6]:
↦    1
```

D.8.3.5 fastelim

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: fastelim(i,p[h,o,a,b,e,m]); i=ideal, p=polynomial; h,o,a,b,e=integers
p: product of variables to be eliminated;
Optional parameters:

- h !=0: use Hilbert-series driven std-basis computation
- o !=0: use proc `valvars` for a - hopefully - optimal ordering of vars
- a !=0: order vars to be eliminated w.r.t. increasing complexity
- b !=0: order vars not to be eliminated w.r.t. increasing complexity
- e !=0: use `elimpart` first to eliminate easy part
- m !=0: compute a minimal system of generators

(default: h,o,a,b,e,m = 0,1,0,0,0,0)

Return: ideal obtained from i by eliminating those variables, which occur in p

Example:

```
LIB "presolve.lib";
ring s=31991,(e,f,x,y,z,t,u,v,w,a,b,c,d),dp;
ideal i = w2+f2-1, x2+t2+a2-1,  y2+u2+b2-1, z2+v2+c2-1,
d2+e2-1, f4+2u, wa+tf, xy+tu+ab;
fastelim(i,xytua,1,1);       //with hilb,valvars
↦ _[1]=z2+v2+c2-1
↦ _[2]=f2+w2-1
↦ _[3]=e2+d2-1
fastelim(i,xytua,1,0,1);     //with hilb,minbase
↦ _[1]=z2+v2+c2-1
↦ _[2]=f2+w2-1
↦ _[3]=e2+d2-1
```

D.8.3.6 findvars

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: findvars(id); id=poly/ideal/vector/module/matrix

Return: list L with 4 entries:

L[1]: ideal of variables occurring in id
L[2]: intvec of variables occurring in id
L[3]: ideal of variables not occurring in id
L[4]: intvec of variables not occurring in id

Example:

```
LIB "presolve.lib";
ring s  = 0,(e,f,x,y,t,u,v,w,a,d),dp;
ideal i = w2+f2-1, x2+t2+a2-1;
findvars(i);
↦ [1]:
↦    _[1]=f
↦    _[2]=x
↦    _[3]=t
↦    _[4]=w
↦    _[5]=a
↦ [2]:
↦    2,3,5,8,9
↦ [3]:
↦    _[1]=e
↦    _[2]=y
↦    _[3]=u
↦    _[4]=v
↦    _[5]=d
↦ [4]:
↦    1,4,6,7,10
```

See also: Section 5.1.166 [variables], page 285.

D.8.3.7 hilbvec

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: hilbvec(id[,c,o]); id=poly/ideal/vector/module/matrix, c,o=strings, c=char, o=ordering used by `hilb` (default: c="32003", o="dp")

Return: bigintvec of 1st Hilbert-series of id, computed in char c and ordering o

Note: id must be homogeneous (i.e. all vars have weight 1)

Example:

```
LIB "presolve.lib";
ring s   = 0,(e,f,x,y,z,t,u,v,w,a,b,c,d,H),dp;
ideal id = w2+f2-1, x2+t2+a2-1,  y2+u2+b2-1, z2+v2+c2-1,
d2+e2-1, f4+2u, wa+tf, xy+tu+ab;
id = homog(id,H);
hilbvec(id);
↦ 1,0,-7,0,20,0,-28,0,14,0,14,0,-28,0,20,0,-7,0,1,0
```

D.8.3.8 linearpart

Procedure from library presolve.lib (see Section D.8.3 [presolve_lib], page 1828).

Usage: linearpart(id); id=ideal/module

Return: list of size 2,
_[1]: generators of id of total degree <= 1
_[2]: remaining generators of id

Note: all variables have degree 1 (independent of ordering of basering)

Example:

```
LIB "presolve.lib";
ring r=0,(x,y,z),dp;
ideal i=1+x+x2+x3,3,x+3y+5z;
linearpart(i);
↦ [1]:
↦    _[1]=3
↦    _[2]=x+3y+5z
↦ [2]:
↦    _[1]=x3+x2+x+1
module m=[x,y,z],x*[x3,y2,z],[1,x2,z3,0,1];
show(linearpart(m));
↦ // list, 2 element(s):
↦ [1]:
↦    // module, 1 generator(s)
↦ [x,y,z]
↦ [2]:
↦    // module, 2 generator(s)
↦ [x4,xy2,xz]
↦ [1,x2,z3,0,1]
```

D.8.3.9 tolessvars

Procedure from library presolve.lib (see Section D.8.3 [presolve_lib], page 1828).

Usage: tolessvars(id [,s1,s2]); id poly/ideal/vector/module/matrix, s1=string (new ordering) [default: s1="dp" or "ds" depending on whether the first block of the old ordering is a p- or an s-ordering, respectively]

Return: If id contains all vars of the basering: empty list.
Else: ring R with the same char as the basering, but possibly less variables (only those variables which actually occur in id). In R an object IMAG (image of id under imap) is stored.

Display: If printlevel >=0, display ideal of vars, which have been omitted from the old ring.

Example:

```
LIB "presolve.lib";
ring r  = 0,(x,y,z),dp;
ideal i = y2-x3,x-3,y-2x;
def R_r = tolessvars(i,"lp");
↦
↦ // variables which did not occur:
↦ z
↦
↦ // 'tolessvars' created a ring, in which an object IMAG is stored.
↦ // To access the object, type (if the name R was assigned to the return v\
   alue):
↦         setring R; IMAG;
setring R_r;
show(basering);
↦ // ring: (QQ),(x,y),(lp(2),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦ // IMAG                           [0]  ideal, 3 generator(s)
IMAG;
↦ IMAG[1]=-x3+y2
↦ IMAG[2]=x-3
↦ IMAG[3]=-2x+y
kill R_r;
```

D.8.3.10 solvelinearpart

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: solvelinearpart(id [,n]); id=ideal/module, n=integer (default: n=0)

Return: (interreduced) generators of id of degree <=1 in reduced triangular form if n=0 [non-reduced triangular form if n!=0]

Assume: monomial ordering is a global ordering (p-ordering)

Note: may be used to solve a system of linear equations, see `gauss_row` from 'matrix.lib' for a different method

Warning: the result is very likely to be false for 'real' coefficients, use char 0 instead!

Example:

```
LIB "presolve.lib";
// Solve the system of linear equations:
//         3x +   y +  z -  u = 2
//         3x +  8y + 6z - 7u = 1
//        14x + 10y + 6z - 7u = 0
//         7x +  4y + 3z - 3u = 3
ring r = 0,(x,y,z,u),lp;
ideal i= 3x +   y +  z -  u,
13x +  8y + 6z - 7u,
14x + 10y + 6z - 7u,
7x +  4y + 3z - 3u;
ideal j= 2,1,0,3;
```

```
j = matrix(i)-matrix(j);        // difference of 1x4 matrices
// compute reduced triangular form, setting
solvelinearpart(j);             // the RHS equal 0 gives the solutions!
↦ _[1]=u-4
↦ _[2]=z-4
↦ _[3]=y+1
↦ _[4]=x-1
solvelinearpart(j,1); "";       // triangular form, not reduced
↦ _[1]=u-4
↦ _[2]=2z-u-4
↦ _[3]=11y+5z-8u+23
↦ _[4]=3x+y+z-u-2
↦
```

D.8.3.11 sortandmap

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: sortandmap(id [,n1,p1,n2,p2...,o1,m1,o2,m2...]);
id=poly/ideal/vector/module,
p1,p2,...= polynomials (product of variables),
n1,n2,...= integers,
o1,o2,...= strings,
m1,m2,...= integers
(default: p1=product of all vars, n1=0, o1="dp",m1=0)
the last pi (containing the remaining vars) may be omitted

Return: a ring R, in which a poly/ideal/vector/module IMAG is stored:
- the ring R differs from the active basering only in the choice of monomial ordering and in the sorting of the variables.
- IMAG is the image (under imap) of the input ideal/module id
The new monomial ordering and sorting of vars is as follows:

- each block of vars occurring in pi is sorted w.r.t. its complexity in id,
- ni controls the sorting in i-th block (= vars occurring in pi):
  ni=0 (resp. ni!=0) means that least complex (resp. most complex) vars come first
- oi and mi define the monomial ordering of the i-th block:
  if mi =0, oi=ordstr(i-th block)
  if mi!=0, the ordering of the i-th block itself is a blockordering,
    each subblock having ordstr=oi, such that vars of same complexity are
    in one block

Note that only simple ordstrings oi are allowed: "lp","dp","Dp", "ls","ds","Ds".

Note: We define a variable x to be more complex than y (with respect to id) if val(x) > val(y) lexicographically, where val(x) denotes the valuation vector of x:
consider id as list of polynomials in x with coefficients in the remaining variables. Then:
val(x) = (maximal occurring power of x, # of all monomials in leading coefficient, # of all monomials in coefficient of next smaller power of x,...).

Example:

```
LIB "presolve.lib";
```

```
ring s = 32003,(x,y,z),dp;
ideal i=x3+y2,xz+z2;
def R_r=sortandmap(i);
↦
↦ // 'sortandmap' created a ring, in which an object IMAG is stored.
↦ // To access the object, type (if the name R was assigned to the return v\
   alue):
↦         setring R; IMAG;
show(R_r);
↦ // ring: (ZZ/32003),(x,y,z),(dp(3),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦ // IMAG                           [0]  ideal, 2 generator(s)
setring R_r; IMAG;
↦ IMAG[1]=x3+y2
↦ IMAG[2]=xz+z2
kill R_r; setring s;
def R_r=sortandmap(i,1,xy,0,z,0,"ds",0,"lp",0);
↦
↦ // 'sortandmap' created a ring, in which an object IMAG is stored.
↦ // To access the object, type (if the name R was assigned to the return v\
   alue):
↦         setring R; IMAG;
show(R_r);
↦ // ring: (ZZ/32003),(x,y,z),(ds(2),lp(1),C);
↦ // minpoly = 0
↦ // objects belonging to this ring:
↦ // IMAG                           [0]  ideal, 2 generator(s)
setring R_r; IMAG;
↦ IMAG[1]=y2+x3
↦ IMAG[2]=z2+xz
kill R_r;
```

D.8.3.12 sortvars

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: sortvars(id[,n1,p1,n2,p2,...]);
id=poly/ideal/vector/module,
p1,p2,...= polynomials (product of vars),
n1,n2,...= integers
(default: p1=product of all vars, n1=0)
the last pi (containing the remaining vars) may be omitted

Compute: sort variables with respect to their complexity in id

Return: list of two elements, an ideal and a list:
[1]: ideal, variables of basering sorted w.r.t their complexity in id
ni controls the ordering in i-th block (= vars occurring in pi):
ni=0 (resp. ni!=0) means that less (resp. more) complex vars come first
[2]: a list with 4 entries for each pi:
_[1]: ideal ai : vars of pi in correct order,
_[2]: intvec vi: permutation vector describing the ordering in ai,
_[3]: intmat Mi: valuation matrix of ai, the columns of Mi being the

valuation vectors of the vars in ai
_[4]: intvec wi: size of 1-st, 2-nd,... block of identical columns of Mi
(vars with same valuation)

Note: We define a variable x to be more complex than y (with respect to id) if val(x) > val(y) lexicographically, where val(x) denotes the valuation vector of x:
consider id as list of polynomials in x with coefficients in the remaining variables. Then:
val(x) = (maximal occurring power of x, # of all monomials in leading coefficient, # of all monomials in coefficient of next smaller power of x,...).

Example:

```
LIB "presolve.lib";
ring s=0,(x,y,z,w),dp;
ideal i = x3+y2+yw2,xz+z2,xyz-w2;
sortvars(i,0,xy,1,zw);
↦ [1]:
↦    _[1]=x
↦    _[2]=y
↦    _[3]=z
↦    _[4]=w
↦ [2]:
↦    [1]:
↦       _[1]=y
↦       _[2]=x
↦    [2]:
↦       2,1
↦    [3]:
↦       2,3,
↦       1,1,
↦       2,0,
↦       0,2
↦    [4]:
↦       1,1
↦    [5]:
↦       _[1]=w
↦       _[2]=z
↦    [6]:
↦       2,1
↦    [7]:
↦       2,2,
↦       2,1,
↦       0,2
↦    [8]:
↦       1,1
```

D.8.3.13 valvars

Procedure from library presolve.lib (see Section D.8.3 [presolve_lib], page 1828).

Usage: valvars(id[,n1,p1,n2,p2,...]);
id=poly/ideal/vector/module,
p1,p2,...= polynomials (product of vars),
n1,n2,...= integers,

ni controls the ordering of vars occurring in pi: ni=0 (resp. ni!=0) means that less (resp. more) complex vars come first (default: p1=product of all vars, n1=0),
the last pi (containing the remaining vars) may be omitted

Compute: valuation (complexity) of variables with respect to id.
ni controls the ordering of vars occurring in pi:
ni=0 (resp. ni!=0) means that less (resp. more) complex vars come first.

Return: list with 3 entries:

```
[1]: intvec, say v, describing the permutation such that the permuted
     ring variables are ordered with respect to their complexity in id
[2]: list of intvecs, i-th intvec, say v(i) describing permutation
     of vars in a(i) such that v=v(1),v(2),...
[3]: list of ideals and intmat's, say a(i) and M(i), where
     a(i): factors of pi,
     M(i): valuation matrix of a(i), such that the j-th column of M(i)
          is the valuation vector of j-th generator of a(i)
```

Note: Use `sortvars` in order to actually sort the variables! We define a variable x to be more complex than y (with respect to id) if val(x) > val(y) lexicographically, where val(x) denotes the valuation vector of x:
consider id as list of polynomials in x with coefficients in the remaining variables. Then:
val(x) = (maximal occurring power of x, # of all monomials in leading coefficient, # of all monomials in coefficient of next smaller power of x,...).

Example:

```
LIB "presolve.lib";
ring s=0,(x,y,z,a,b),dp;
ideal i=ax2+ay3-b2x,abz+by2;
valvars (i,0,xyz);
↦ [1]:
↦    1,2,3,4,5
↦ [2]:
↦    [1]:
↦       3,1,2
↦    [2]:
↦       1,2
↦ [3]:
↦    [1]:
↦       _[1]=x
↦       _[2]=y
↦       _[3]=z
↦    [2]:
↦       2,3,1,
↦       1,1,1,
↦       1,1,0
↦    [3]:
↦       _[1]=a
↦       _[2]=b
↦    [4]:
↦       1,2,
↦       3,1,
↦       0,2
```

D.8.3.14 idealSplit

Procedure from library `presolve.lib` (see Section D.8.3 [presolve_lib], page 1828).

Usage: idealSplit(id,timeF,timeS); id ideal and optional
timeF, timeS integers to bound the time which can be used for factorization resp. standard basis computation

Return: a list of ideals such that their intersection
has the same radical as id

Example:

```
LIB "presolve.lib";
ring r=32003,(b,s,t,u,v,w,x,y,z),dp;
ideal i=
bv+su,
bw+tu,
sw+tv,
by+sx,
bz+tx,
sz+ty,
uy+vx,
uz+wx,
vz+wy,
bvz;
idealSplit(i);
↦ [1]:
↦    _[1]=x
↦    _[2]=u
↦    _[3]=t
↦    _[4]=s
↦    _[5]=b
↦    _[6]=wy+vz
↦ [2]:
↦    _[1]=z
↦    _[2]=w
↦    _[3]=t
↦    _[4]=s
↦    _[5]=b
↦    _[6]=vx+uy
↦ [3]:
↦    _[1]=z
↦    _[2]=x
↦    _[3]=w
↦    _[4]=u
↦    _[5]=t
↦    _[6]=b
↦ [4]:
↦    _[1]=z
↦    _[2]=y
↦    _[3]=x
↦    _[4]=t
↦    _[5]=s
↦    _[6]=b
↦ [5]:
↦    _[1]=z
↦    _[2]=y
↦    _[3]=x
↦    _[4]=u
↦    _[5]=b
↦    _[6]=tv+sw
↦ [6]:
↦    _[1]=z
↦    _[2]=y
↦    _[3]=x
↦    _[4]=w
↦    _[5]=t
↦    _[6]=su+bv
↦ [7]:
↦    _[1]=w
↦    _[2]=v
↦    _[3]=u
↦    _[4]=t
↦    _[5]=s
↦    _[6]=b
↦ [8]:
↦    _[1]=x
↦    _[2]=w
↦    _[3]=v
↦    _[4]=u
↦    _[5]=b
↦    _[6]=ty+sz
↦ [9]:
↦    _[1]=z
↦    _[2]=w
↦    _[3]=v
↦    _[4]=u
↦    _[5]=t
↦    _[6]=sx+by
↦ [10]:
↦    _[1]=z
↦    _[2]=y
↦    _[3]=x
↦    _[4]=w
↦    _[5]=v
↦    _[6]=u
↦ [11]:
↦    _[1]=y
↦    _[2]=v
↦    _[3]=t
↦    _[4]=s
↦    _[5]=b
↦    _[6]=wx+uz
↦ [12]:
↦    _[1]=y
↦    _[2]=x
↦    _[3]=v
↦      _[4]=u
↦      _[5]=s
↦      _[6]=b
↦   [13]:
↦      _[1]=z
↦      _[2]=y
↦      _[3]=x
↦      _[4]=v
↦      _[5]=s
↦      _[6]=tu+bw
↦   [14]:
↦      _[1]=z
↦      _[2]=y
↦      _[3]=w
↦      _[4]=v
↦      _[5]=t
↦      _[6]=s
↦   [15]:
↦      _[1]=y
↦      _[2]=w
↦      _[3]=v
↦      _[4]=u
↦      _[5]=s
↦      _[6]=tx+bz
```

D.8.4 solve_lib

Library: solve.lib

Purpose: Complex Solving of Polynomial Systems

Author: Moritz Wenk, email: wenk@mathematik.uni-kl.de
Wilfred Pohl, email: pohl@mathematik.uni-kl.de

Procedures:

D.8.4.1 laguerre_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: laguerre_solve(f [, m, l, n, s]); f = polynomial,
m, l, n, s = integers (control parameters of the method)
m: precision of output in digits (4 <= m), if basering is not ring of complex numbers;
l: precision of internal computation in decimal digits (l >=8) only if the basering is not complex or complex with smaller precision;
n: control of multiplicity of roots or of splitting of f into squarefree factors
n < 0, no split of f (good, if all roots are simple)
n >= 0, try to split
n = 0, return only different roots
n > 0, find all roots (with multiplicity)
s: s != 0, returns ERROR if | f(root) | > 0.1^m (when computing in the current ring)
(default: m, l, n, s = 8, 30, 1, 0)

Assume: f is a univariate polynomial;
basering has characteristic 0 and is either complex or without parameters.

Return: list of (complex) roots of the polynomial f, depending on n. The entries of the result are of type
string: if the basering is not complex,
number: otherwise.

Note: If printlevel >0: displays comments (default = 0).
If s != 0 and if the procedure stops with ERROR, try a higher internal precision m.

Example:

```
LIB "solve.lib";
// Find all roots of an univariate polynomial using Laguerre's method:
ring rs1= 0,(x,y),lp;
poly f = 15x5 + x3 + x2 - 10;
// 10 digits precision
laguerre_solve(f,10);
↦ [1]:
↦    0.8924637479
↦ [2]:
↦    (-0.7392783383+i*0.5355190078)
↦ [3]:
↦    (-0.7392783383-i*0.5355190078)
↦ [4]:
↦    (0.2930464644-i*0.9003002396)
↦ [5]:
↦    (0.2930464644+i*0.9003002396)
// Now with complex coefficients,
// internal precision is 30 digits (default)
printlevel=2;
ring rsc= (real,10,i),x,lp;
poly f = (15.4+i*5)*x^5 + (25.0e-2+i*2)*x^3 + x2 - 10*i;
list l = laguerre_solve(f);
↦ //BEGIN laguerre_solve
↦ //control: complex ring with precision 30
↦ //working in:  complex,10,10,i
↦ //         polynomial has complex coefficients
↦ //split in working ring:
↦ //split without result
↦ //END laguerre_solve
l;
↦ [1]:
↦ (-0.8557376852+i*0.3557664188)
↦ [2]:
↦ (-0.5462895588-i*0.6796668873)
↦ [3]:
↦ (0.04588498039+i*0.9133296179)
↦ [4]:
↦ (0.5037408279-i*0.8058051828)
↦ [5]:
↦ (0.8524014357+i*0.2163760334)
// check result, value of substituted polynomial should be near to zero
// remember that l contains a list of strings
// in the case of a different ring
subst(f,x,l[1]);
↦ 0
```

```
subst(f,x,l[2]);
↦ 0
```

D.8.4.2 solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: solve(G [, m, n [, l]] [,"oldring"] [,"nodisplay"]); G = ideal, m, n, l = integers (control parameters of the method), outR ring,
m: precision of output in digits (4 <= m) and of the generated ring of complex numbers;
n: control of multiplicity
n = 0, return all different roots
n != 0, find all roots (with multiplicity)
l: precision of internal computation in decimal digits (l >=8) only if the basering is not complex or complex with smaller precision,
[default: (m,n,l) = (8,0,30), or if only (m,n) are set explicitly with n!=0, then (m,n,l) = (m,n,60)]

Assume: the ideal is 0-dimensional;
basering has characteristic 0 and is either complex or without parameters;

Return: (1) If called without the additional parameter `"oldring"`:
ring `R` with the same number of variables but with complex coefficients (and precision m). `R` comes with a list `SOL` of numbers, in which complex roots of G are stored:
* If n = 0, `SOL` is the list of all different solutions, each of them being represented by a list of numbers.
* If n != 0, `SOL` is a list of two list: SOL[i][1] is the list of all different solutions with the multiplicity SOL[i][2].
SOL is ordered w.r.t. multiplicity (the smallest first).
(2) If called with the additional parameter `"oldring"`, the procedure looks for an appropriate ring (at top level) in which the solutions can be stored (interactive).
The user may then select an appropriate ring and choose a name for the output list in this ring. The list is exported directly to the selected ring and the return value is a string "result exported to" + name of the selected ring.

Note: If the problem is not 0-dim. the procedure stops with ERROR. If the ideal G is not a lexicographic Groebner basis, the lexicographic Groebner basis is computed internally (Hilbert driven).
The computed solutions are displayed, unless `solve` is called with the additional parameter `"nodisplay"`.

Example:

```
LIB "solve.lib";
// Find all roots of a multivariate ideal using triangular sets:
int d,t,s = 4,3,2 ;
int i;
ring A=0,x(1..d),dp;
poly p=-1;
for (i=d; i>0; i--) { p=p+x(i)^s; }
ideal I = x(d)^t-x(d)^s+p;
for (i=d-1; i>0; i--) { I=x(i)^t-x(i)^s+p,I; }
I;
↦ I[1]=x(1)^3+x(2)^2+x(3)^2+x(4)^2-1
↦ I[2]=x(2)^3+x(1)^2+x(3)^2+x(4)^2-1
↦ I[3]=x(3)^3+x(1)^2+x(2)^2+x(4)^2-1
↦ I[4]=x(4)^3+x(1)^2+x(2)^2+x(3)^2-1
// the multiplicity is
vdim(std(I));
↦ 81
def AC=solve(I,6,0,"nodisplay");  // solutions should not be displayed
↦ 
↦ // 'solve' created a ring, in which a list SOL of numbers (the complex so\
  lutions)
↦ // is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
  ned
↦ // to the return value):
↦         setring R; SOL;
// list of solutions is stored in AC as the list SOL (default name)
setring AC;
size(SOL);               // number of different solutions
↦ 37
SOL[5];                  // the 5th solution
↦ [1]:
↦ 0.587401
↦ [2]:
↦ -0.32748
↦ [3]:
↦ 0.587401
↦ [4]:
↦ 0.587401
// you must start with char. 0
setring A;
def AC1=solve(I,6,1,"nodisplay");
↦ 
↦ // 'solve' created a ring, in which a list SOL of numbers (the complex so\
  lutions)
↦ // is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
  ned
↦ // to the return value):
↦         setring R; SOL;
setring AC1;
size(SOL);               // number of different multiplicities
↦ 2
SOL[1][1][1];            // a solution with
↦ [1]:
↦ (0.766044+i*0.477895)
↦ [2]:
↦ (0.766044+i*0.477895)
↦ [3]:
↦ (0.766044-i*0.477895)
↦ [4]:
↦ (0.766044-i*0.477895)
SOL[1][2];               // multiplicity 1
↦ 1
```

```
SOL[2][1][1];              // a solution with
↦ [1]:
↦ 0
↦ [2]:
↦ 0
↦ [3]:
↦ 1
↦ [4]:
↦ 0
SOL[2][2];                 // multiplicity 12
↦ 12
// the number of different solutions is equal to
size(SOL[1][1])+size(SOL[2][1]);
↦ 37
// the number of complex solutions (counted with multiplicities) is
size(SOL[1][1])*SOL[1][2]+size(SOL[2][1])*SOL[2][2];
↦ 81
```

D.8.4.3 ures_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: ures_solve(i [, k, p]); i = ideal, k, p = integers
k=0: use sparse resultant matrix of Gelfand, Kapranov and Zelevinsky,
k=1: use resultant matrix of Macaulay which works only for homogeneous ideals,
p>0: defines precision of the long floats for internal computation if the basering is not complex (in decimal digits),
(default: k=0, p=30)

Assume: i is a zerodimensional ideal given by a quadratic system, that is,
nvars(basering) = ncols(i) = number of vars actually occurring in i,

Return: If the ground field is the field of complex numbers: list of numbers (the complex roots of the polynomial system i=0).
Otherwise: ring `R` with the same number of variables but with complex coefficients (and precision p). `R` comes with a list `SOL` of numbers, in which complex roots of the polynomial system i are stored:

Example:

```
LIB "solve.lib";
// compute the intersection points of two curves
ring rsq = 0,(x,y),lp;
ideal gls=  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R=ures_solve(gls,0,16);
↦
↦ // 'ures_solve' created a ring, in which a list SOL of numbers (the compl\
   ex
↦ // solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; SOL;
setring R; SOL;
```

```
↦ [1]:
↦    [1]:
↦ -2.82842712474619
↦    [2]:
↦ -1.414213562373095
↦ [2]:
↦    [1]:
↦       -1
↦    [2]:
↦       3
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       -3
↦ [4]:
↦    [1]:
↦ 2.82842712474619
↦    [2]:
↦ 1.414213562373095
```

D.8.4.4 mp_res_mat

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: mp_res_mat(i [, k]); i ideal, k integer,
k=0: sparse resultant matrix of Gelfand, Kapranov and Zelevinsky,
k=1: resultant matrix of Macaulay (k=0 is default)

Assume: The number of elements in the input system must be the number of variables in the basering plus one;
if k=1 then i must be homogeneous.

Return: module representing the multipolynomial resultant matrix

Example:

```
LIB "solve.lib";
// compute resultant matrix in ring with parameters (sparse resultant matrix)
ring rsq= (0,u0,u1,u2),(x1,x2),lp;
ideal i= u0+u1*x1+u2*x2,x1^2 + x2^2 - 10,x1^2 + x1*x2 + 2*x2^2 - 16;
module m = mp_res_mat(i);
print(m);
↦ -16,0,  -10,0,  (u0),0,   0,  0,   0,   0,
↦ 0,  -16,0,  -10,(u2),(u0),0,  0,   0,   0,
↦ 2,  0,  1,  0,  0,   (u2),0,  0,   0,   0,
↦ 0,  2,  0,  1,  0,   0,   0,  0,   0,   0,
↦ 0,  0,  0,  0,  (u1),0,   -10,(u0),0,   -16,
↦ 1,  0,  0,  0,  0,   (u1),0,  (u2),(u0),0,
↦ 0,  1,  0,  0,  0,   0,   1,  0,   (u2),2,
↦ 1,  0,  1,  0,  0,   0,   0,  (u1),0,   0,
↦ 0,  1,  0,  1,  0,   0,   0,  0,   (u1),1,
↦ 0,  0,  0,  0,  0,   0,   1,  0,   0,   1
// computing sparse resultant
det(m);
↦ (-2*u0^4+18*u0^2*u1^2+4*u0^2*u1*u2+22*u0^2*u2^2-16*u1^4+80*u1^3*u2-52*u1^\
   2*u2^2-120*u1*u2^3-36*u2^4)
// compute resultant matrix (Macaulay resultant matrix)
ring rdq= (0,u0,u1,u2),(x0,x1,x2),lp;
ideal h=  homog(imap(rsq,i),x0);
h;
↦ h[1]=(u0)*x0+(u1)*x1+(u2)*x2
↦ h[2]=-10*x0^2+x1^2+x2^2
↦ h[3]=-16*x0^2+x1^2+x1*x2+2*x2^2
module m = mp_res_mat(h,1);
print(m);
↦ x0, x1, x2, 0, 0, 0, 0,0, 0, 0,
↦ 0,  x0, 0,  x1,x2,0, 0,0, 0, 0,
↦ 0,  0,  x0, 0, x1,x2,0,0, 0, 0,
↦ -10,0,  0,  1, 0, 1, 0,0, 0, 0,
↦ 0,  0,  0,  0, x0,0, 0,x1,x2,0,
↦ -16,0,  0,  1, 1, 2, 0,0, 0, 0,
↦ 0,  -10,0,  0, 0, 0, 1,0, 1, 0,
↦ 0,  0,  -10,0, 0, 0, 0,1, 0, 1,
↦ 0,  -16,0,  0, 0, 0, 1,1, 2, 0,
↦ 0,  0,  -16,0, 0, 0, 0,1, 1, 2
// computing Macaulay resultant (should be the same as above!)
det(m);
↦ 2*x0^4-18*x0^2*x1^2-4*x0^2*x1*x2-22*x0^2*x2^2+16*x1^4-80*x1^3*x2+52*x1^2*\
   x2^2+120*x1*x2^3+36*x2^4
// compute numerical sparse resultant matrix
setring rsq;
ideal ir= 15+2*x1+5*x2,x1^2 + x2^2 - 10,x1^2 + x1*x2 + 2*x2^2 - 16;
module mn = mp_res_mat(ir);
print(mn);
↦ 15,0, -10,0,  0, 0, 0,  -16,0,  0,
↦ 5, 15,0,  -10,0, 0, 0,  0,  -16,0,
↦ 0, 5, 1,  0,  0, 0, 0,  2,  0,  0,
↦ 0, 0, 0,  1,  0, 0, 0,  0,  2,  0,
↦ 2, 0, 0,  0,  15,0, -10,0,  0,  -16,
↦ 0, 2, 0,  0,  5, 15,0,  1,  0,  0,
↦ 0, 0, 0,  0,  0, 5, 1,  0,  1,  2,
↦ 0, 0, 1,  0,  2, 0, 0,  1,  0,  0,
↦ 0, 0, 0,  1,  0, 2, 0,  0,  1,  1,
↦ 0, 0, 0,  0,  0, 0, 1,  0,  0,  1
// computing sparse resultant
det(mn);
↦ -7056
```

D.8.4.5 interpolate

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: interpolate(p,v,d); p,v=ideals of numbers, d=integer

Assume: Ground field K is the field of rational numbers, p and v are lists of elements of the ground field K with p[j] != -1,0,1, size(p) = n (= number of vars) and size(v)=N=(d+1)^n.

Return: poly f, the unique polynomial f of degree n*d with prescribed values v[i] at the points p(i)=(p[1]^(i-1),..,p[n]^(i-1)), i=1,..,N.

Note: mainly useful when n=1, i.e. f is satisfying f(p^(i-1)) = v[i], i=1..d+1.

Example:

```
LIB "solve.lib";
ring r1 = 0,(x),lp;
// determine f with deg(f) = 4 and
// v = values of f at points 3^0, 3^1, 3^2, 3^3, 3^4
ideal v=16,0,11376,1046880,85949136;
interpolate( 3, v, 4 );
↦ 2x4-22x2+36
```

See also: Section 5.1.164 [vandermonde], page 284.

D.8.4.6 fglm_solve

Procedure from library solve.lib (see Section D.8.4 [solve_lib], page 1842).

Usage: fglm_solve(i [, p]); i ideal, p integer

Assume: the ground field has char 0.

Return: ring R with the same number of variables but with complex coefficients (and precision p). R comes with a list rlist of numbers, in which the complex roots of i are stored. p>0: gives precision of complex numbers in decimal digits [default: p=30].

Note: The procedure uses a standard basis of i to determine all complex roots of i.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),dp;
// compute the intersection points of two curves
ideal s =  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R = fglm_solve(s,10);
↦
↦ // 'fglm_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [2]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [3]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
↦ [4]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
```

D.8.4.7 lex_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: lex_solve(i[,p]); i=ideal, p=integer,
p>0: gives precision of complex numbers in decimal digits (default: p=30).

Assume: i is a reduced lexicographical Groebner bases of a zero-dimensional ideal, sorted by increasing leading terms.

Return: ring `R` with the same number of variables but with complex coefficients (and precision p). `R` comes with a list `rlist` of numbers, in which the complex roots of i are stored.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),lp;
// compute the intersection points of two curves
ideal s =  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R = lex_solve(stdfglm(s),10);
↦
↦ // 'lex_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [2]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [3]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
↦ [4]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
```

D.8.4.8 simplexOut

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: simplexOut(l); l list

Assume: l is the output of simplex.

Return: Nothing. The procedure prints the computed solution of simplex (as strings) in a nice format.

Example:

```
LIB "solve.lib";
ring r = (real,10),(x),lp;
// consider the max. problem:
//
//    maximize  x(1) + x(2) + 3*x(3) - 0.5*x(4)
//
//  with constraints:   x(1) +          2*x(3)          <= 740
//                             2*x(2)          - 7*x(4) <=   0
//                               x(2) -   x(3) + 2*x(4) >=   0.5
//                      x(1) +   x(2) +   x(3) +   x(4)  =   9
//
matrix sm[5][5]=   0, 1, 1, 3,-0.5,
740,-1, 0,-2, 0,
0, 0,-2, 0, 7,
0.5, 0,-1, 1,-2,
9,-1,-1,-1,-1;
int n = 4;  // number of constraints
int m = 4;  // number of variables
int m1= 2;  // number of <= constraints
int m2= 1;  // number of >= constraints
int m3= 1;  // number of == constraints
list sol=simplex(sm, n, m, m1, m2, m3);
simplexOut(sol);
↦ z = 17.025
↦ x2 = 3.325
↦ x4 = 0.95
↦ x3 = 4.725
```

See also: Section 5.1.142 [simplex], page 261.

D.8.4.9 triangLf_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: triangLf_solve(i [, p]); i ideal, p integer,
p>0: gives precision of complex numbers in digits (default: p=30).

Assume: the ground field has char 0; i is a zero-dimensional ideal

Return: ring `R` with the same number of variables but with complex coefficients (and precision p). `R` comes with a list `rlist` of numbers, in which the complex roots of i are stored.

Note: The procedure uses a triangular system (Lazard's Algorithm with factorization) computed from a standard basis to determine recursively all complex roots of the input ideal i with Laguerre's algorithm.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),lp;
// compute the intersection points of two curves
ideal s = x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R = triangLf_solve(s,10);
↦
↦ // 'triangLf_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [3]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [4]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
```

D.8.4.10 triangM_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: triangM_solve(i [, p]); i=ideal, p=integer,
p>0: gives precision of complex numbers in digits (default: p=30).

Assume: the ground field has char 0;
i zero-dimensional ideal

Return: ring `R` with the same number of variables but with complex coefficients (and precision p). `R` comes with a list `rlist` of numbers, in which the complex roots of i are stored.

Note: The procedure uses a triangular system (Moellers Algorithm) computed from a standard basis of input ideal i to determine recursively all complex roots with Laguerre's algorithm.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),lp;
```

```
// compute the intersection points of two curves
ideal s =  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R = triangM_solve(s,10);
↦
↦ // 'triangM_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [2]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [3]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
↦ [4]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
```

D.8.4.11 triangL_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: triangL_solve(i [, p]); i=ideal, p=integer,
p>0: gives precision of complex numbers in digits (default: p=30).

Assume: the ground field has char 0; i is a zero-dimensional ideal.

Return: ring `R` with the same number of variables, but with complex coefficients (and precision p). `R` comes with a list `rlist` of numbers, in which the complex roots of i are stored.

Note: The procedure uses a triangular system (Lazard's Algorithm) computed from a standard basis of input ideal i to determine recursively all complex roots with Laguerre's algorithm.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),lp;
// compute the intersection points of two curves
ideal s =  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R = triangL_solve(s,10);
↦
```

```
↦ // 'triangL_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [2]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [3]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
↦ [4]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
```

D.8.4.12 triang_solve

Procedure from library `solve.lib` (see Section D.8.4 [solve_lib], page 1842).

Usage: triang_solve(l,p [,d]); l=list, p,d=integers
l is a list of finitely many triangular systems, such that the union of their varieties equals the variety of the initial ideal.
p>0: gives precision of complex numbers in digits,
d>0: gives precision (1<d<p) for near-zero-determination,
(default: d=1/2*p).

Assume: the ground field has char 0;
l was computed using the algorithm of Lazard or the algorithm of Moeller (see triang.lib).

Return: ring `R` with the same number of variables, but with complex coefficients (and precision p). `R` comes with a list `rlist` of numbers, in which the complex roots of l are stored.

Example:

```
LIB "solve.lib";
ring r = 0,(x,y),lp;
// compute the intersection points of two curves
ideal s=  x2 + y2 - 10, x2 + xy + 2y2 - 16;
def R=triang_solve(triangLfak(stdfglm(s)),10);
↦
```

```
↦ // 'triang_solve' created a ring, in which a list rlist of numbers (the
↦ // complex solutions) is stored.
↦ // To access the list of complex solutions, type (if the name R was assig\
   ned
↦ // to the return value):
↦         setring R; rlist;
setring R; rlist;
↦ [1]:
↦    [1]:
↦ -1
↦    [2]:
↦ 3
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ -3
↦ [3]:
↦    [1]:
↦ -2.828427125
↦    [2]:
↦ -1.414213562
↦ [4]:
↦    [1]:
↦ 2.828427125
↦    [2]:
↦ 1.414213562
```

D.8.5 triang_lib

Library: triang.lib

Purpose: Decompose Zero-dimensional Ideals into Triangular Sets

Author: D. Hillebrand

Procedures:

D.8.5.1 triangL

Procedure from library `triang.lib` (see Section D.8.5 [triang_lib], page 1855).

Usage: triangL(G); G=ideal

Assume: G is the reduced lexicographical Groebner basis of the zero-dimensional ideal (G), sorted by increasing leading terms.

Return: a list of finitely many triangular systems, such that the union of their varieties equals the variety of (G).

Note: Algorithm of Lazard (see: Lazard, D.: Solving zero-dimensional algebraic systems, J. Symb. Comp. 13, 117 - 132, 1992).

Example:

```
LIB "triang.lib";
ring rC5 = 0,(e,d,c,b,a),lp;
```

```
triangL(stdfglm(cyclic(5)));
↦ [1]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c2+3ca+a2
↦    _[4]=d+c+3a
↦    _[5]=e-a
↦ [2]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c-a
↦    _[4]=d2+3da+a2
↦    _[5]=e+d+3a
↦ [3]:
↦    _[1]=a5-1
↦    _[2]=b6+4b5a+5b4a2+5b3a3+5b2a4+4b+a
↦    _[3]=5c+8b5a+30b4a2+30b3a3+25b2a4+30b+22a
↦    _[4]=5d-2b5a-10b4a2-15b3a3-10b2a4-10b-8a
↦    _[5]=5e-6b5a-20b4a2-15b3a3-15b2a4-15b-9a
↦ [4]:
↦    _[1]=a10+123a5+1
↦    _[2]=55b2-2ba6-233ba-8a7-987a2
↦    _[3]=55c+a6+144a
↦    _[4]=55d+a6+144a
↦    _[5]=55e+55b-2a6-233a
```

D.8.5.2 triangLfak

Procedure from library `triang.lib` (see Section D.8.5 [triang_lib], page 1855).

Usage: triangLfak(G); G=ideal

Assume: G is the reduced lexicographical Groebner basis of the zero-dimensional ideal (G), sorted by increasing leading terms.

Return: a list of finitely many triangular systems, such that the union of their varieties equals the variety of (G).

Note: Algorithm of Lazard with factorization (see: Lazard, D.: Solving zero-dimensional algebraic systems, J. Symb. Comp. 13, 117 - 132, 1992).

Remark: each polynomial of the triangular systems is factorized.

Example:

```
LIB "triang.lib";
ring rC5 = 0,(e,d,c,b,a),lp;
triangLfak(stdfglm(cyclic(5)));
↦ [1]:
↦    _[1]=a-1
↦    _[2]=b-1
↦    _[3]=c-1
↦    _[4]=d2+3d+1
↦    _[5]=e+d+3
↦ [2]:
↦    _[1]=a-1
↦    _[2]=b-1
↦      _[3]=c2+3c+1
↦      _[4]=d+c+3
↦      _[5]=e-1
↦   [3]:
↦      _[1]=a-1
↦      _[2]=b2+3b+1
↦      _[3]=c+b+3
↦      _[4]=d-1
↦      _[5]=e-1
↦   [4]:
↦      _[1]=a-1
↦      _[2]=b4+b3+b2+b+1
↦      _[3]=-c+b2
↦      _[4]=-d+b3
↦      _[5]=e+b3+b2+b+1
↦   [5]:
↦      _[1]=a2+3a+1
↦      _[2]=b-1
↦      _[3]=c-1
↦      _[4]=d-1
↦      _[5]=e+a+3
↦   [6]:
↦      _[1]=a2+3a+1
↦      _[2]=b+a+3
↦      _[3]=c-1
↦      _[4]=d-1
↦      _[5]=e-1
↦   [7]:
↦      _[1]=a4-4a3+6a2+a+1
↦      _[2]=-11b2+6ba3-26ba2+41ba-4b-8a3+31a2-40a-24
↦      _[3]=11c+3a3-13a2+26a-2
↦      _[4]=11d+3a3-13a2+26a-2
↦      _[5]=-11e-11b+6a3-26a2+41a-4
↦   [8]:
↦      _[1]=a4+a3+a2+a+1
↦      _[2]=b-1
↦      _[3]=c+a3+a2+a+1
↦      _[4]=-d+a3
↦      _[5]=-e+a2
↦   [9]:
↦      _[1]=a4+a3+a2+a+1
↦      _[2]=b-a
↦      _[3]=c-a
↦      _[4]=d2+3da+a2
↦      _[5]=e+d+3a
↦   [10]:
↦      _[1]=a4+a3+a2+a+1
↦      _[2]=b-a
↦      _[3]=c2+3ca+a2
↦      _[4]=d+c+3a
↦      _[5]=e-a
↦   [11]:
↦      _[1]=a4+a3+a2+a+1
↦    _[2]=b3+b2a+b2+ba2+ba+b+a3+a2+a+1
↦    _[3]=c+b2a3+b2a2+b2a+b2
↦    _[4]=-d+b2a2+b2a+b2+ba2+ba+a2
↦    _[5]=-e+b2a3-ba2-ba-b-a2-a
↦ [12]:
↦    _[1]=a4+a3+a2+a+1
↦    _[2]=b2+3ba+a2
↦    _[3]=c+b+3a
↦    _[4]=d-a
↦    _[5]=e-a
↦ [13]:
↦    _[1]=a4+a3+6a2-4a+1
↦    _[2]=-11b2+6ba3+10ba2+39ba+2b+16a3+23a2+104a-24
↦    _[3]=11c+3a3+5a2+25a+1
↦    _[4]=11d+3a3+5a2+25a+1
↦    _[5]=-11e-11b+6a3+10a2+39a+2
```

D.8.5.3 triangM

Procedure from library `triang.lib` (see Section D.8.5 [triang_lib], page 1855).

Usage: triangM(G[,i]); G=ideal, i=integer,

Assume: G is the reduced lexicographical Groebner basis of the zero-dimensional ideal (G), sorted by increasing leading terms.

Return: a list of finitely many triangular systems, such that the union of their varieties equals the variety of (G). If $i = 2$, then each polynomial of the triangular systems is factorized.

Note: Algorithm of Moeller (see: Moeller, H.M.: On decomposing systems of polynomial equations with finitely many solutions, Appl. Algebra Eng. Commun. Comput. 4, 217 - 230, 1993).

Example:

```
LIB "triang.lib";
ring rC5 = 0,(e,d,c,b,a),lp;
triangM(stdfglm(cyclic(5))); //oder: triangM(stdfglm(cyclic(5)),2);
↦ [1]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c-a
↦    _[4]=d2+3da+a2
↦    _[5]=e+d+3a
↦ [2]:
↦    _[1]=a10+123a5+1
↦    _[2]=55b2-2ba6-233ba-8a7-987a2
↦    _[3]=55c+a6+144a
↦    _[4]=55d+a6+144a
↦    _[5]=55e+55b-2a6-233a
↦ [3]:
↦    _[1]=a5-1
↦    _[2]=b6+4b5a+5b4a2+5b3a3+5b2a4+4b+a
↦    _[3]=5c+8b5a+30b4a2+30b3a3+25b2a4+30b+22a
↦    _[4]=5d-2b5a-10b4a2-15b3a3-10b2a4-10b-8a
↦    _[5]=5e-6b5a-20b4a2-15b3a3-15b2a4-15b-9a
↦ [4]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c2+3ca+a2
↦    _[4]=d+c+3a
↦    _[5]=e-a
```

D.8.5.4 triangMH

Procedure from library `triang.lib` (see Section D.8.5 [triang_lib], page 1855).

Usage: triangMH(G[,i]); G=ideal, i=integer

Assume: G is the reduced lexicographical Groebner basis of the zero-dimensional ideal (G), sorted by increasing leading terms.

Return: a list of finitely many triangular systems, such that the disjoint union of their varieties equals the variety of (G). If i = 2, then each polynomial of the triangular systems is factorized.

Note: Algorithm of Moeller and Hillebrand (see: Moeller, H.M.: On decomposing systems of polynomial equations with finitely many solutions, Appl. Algebra Eng. Commun. Comput. 4, 217 - 230, 1993 and Hillebrand, D.: Triangulierung nulldimensionaler Ideale - Implementierung und Vergleich zweier Algorithmen, master thesis, Universitaet Dortmund, Fachbereich Mathematik, Prof. Dr. H.M. Moeller, 1999).

Example:

```
LIB "triang.lib";
ring rC5 = 0,(e,d,c,b,a),lp;
triangMH(stdfglm(cyclic(5)));
↦ [1]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c-a
↦    _[4]=d2+3da+a2
↦    _[5]=e+d+3a
↦ [2]:
↦    _[1]=a10+123a5+1
↦    _[2]=55b2-2ba6-233ba-8a7-987a2
↦    _[3]=55c+a6+144a
↦    _[4]=55d+a6+144a
↦    _[5]=55e+55b-2a6-233a
↦ [3]:
↦    _[1]=a5-1
↦    _[2]=b6+4b5a+5b4a2+5b3a3+5b2a4+4b+a
↦    _[3]=5c+8b5a+30b4a2+30b3a3+25b2a4+30b+22a
↦    _[4]=5d-2b5a-10b4a2-15b3a3-10b2a4-10b-8a
↦    _[5]=5e-6b5a-20b4a2-15b3a3-15b2a4-15b-9a
↦ [4]:
↦    _[1]=a5-1
↦    _[2]=b-a
↦    _[3]=c2+3ca+a2
↦    _[4]=d+c+3a
↦    _[5]=e-a
```

D.8.6 ntsolve_lib

Library: ntsolve.lib

Purpose: Real Newton Solving of Polynomial Systems

Authors: Wilfred Pohl, email: pohl@mathematik.uni-kl.de
Dietmar Hillebrand

Procedures:

D.8.6.1 nt_solve

Procedure from library `ntsolve.lib` (see Section D.8.6 [ntsolve_lib], page 1860).

Usage: nt_solve(gls,ini[,ipar]); gls,ini= ideals, ipar=list/intvec,
gls: contains the equations, for which a solution will be computed
ini: ideal of initial values (approximate solutions to start with),
ipar: control integers (default: ipar = [100, 10])

ipar[1]: max. number of iterations
ipar[2]: accuracy (we have the l_2-norm ||.||): accepts solution `sol`
if ||gls(sol)|| < eps0*(0.1^ipar[2])
where eps0 = ||gls(ini)|| is the initial error

Assume: gls is a zerodimensional ideal with nvars(basering) = size(gls) (>1)

Return: ideal, coordinates of one solution (if found), 0 else

Note: if printlevel >0: displays comments (default =0)

Example:

```
LIB "ntsolve.lib";
ring rsq = (real,40),(x,y,z,w),lp;
ideal gls =  x2+y2+z2-10, y2+z3+w-8, xy+yz+xz+w5 - 1,w3+y;
ideal ini = 3.1,2.9,1.1,0.5;
intvec ipar = 200,0;
ideal sol = nt_solve(gls,ini,ipar);
sol;
↦ sol[1]=0.8698104581550055082008024755236550190005
↦ sol[2]=2.821577445750324600849626251551515976097
↦ sol[3]=1.132312008466417990006094015504381806805
↦ sol[4]=-1.413071026406678849397999475515923744295
```

D.8.6.2 triMNewton

Procedure from library `ntsolve.lib` (see Section D.8.6 [ntsolve_lib], page 1860).

Usage: triMNewton(G,a[,ipar]); G,a= ideals, ipar=list/intvec

Assume: G: g1,..,gn, a triangular system of n equations in n vars, i.e. gi=gi(var(n-i+1),..,var(n)),
a: ideal of numbers, coordinates of an approximation of a common zero of G to start with (with a[i] to be substituted in var(i)),
ipar: control integer vector (default: ipar = [100, 10])

ipar[1]: max. number of iterations
ipar[2]: accuracy (we have as norm |.| absolute value):
accepts solution `sol` if |G(sol)| < |G(a)|*(0.1^ipar[2]).

Return: an ideal, coordinates of a better approximation of a zero of G

Example:

```
LIB "ntsolve.lib";
ring r = (real,30),(z,y,x),(lp);
ideal i = x^2-1,y^2+x4-3,z2-y4+x-1;
ideal a = 2,3,4;
intvec e = 20,10;
ideal l = triMNewton(i,a,e);
l;
↦ l[1]=-2.00000000004226573888027914342
↦ l[2]=1.41421356237309504880168872421
↦ l[3]=1
```

D.8.7 recover_lib

Library: recover.lib

Purpose: Hybrid numerical/symbolical algorithms for algebraic geometry

Author: Adrian Koch (kocha at rhrk.uni-kl.de)

Overview: In this library you'll find implementations of some of the algorithms presented in the paper listed below: Bertini is used to compute a witness set of a given ideal I. Then a lattice basis reduction algorithm is used to recover exact results from the inexact numerical data. More precisely, we obtain elements of prime components of I, the radical of I, or an elimination ideal of I.

NOTE that Bertini may create quite a lot of files in the current directory (or overwrite files which have the same names as the files it wants to create). It also prints information to the screen.

The usefulness of the results of the exactness recovery algorithms heavily depends on the quality of the witness set and the quality of the lattice basis reduction algorithm. The procedures requiring a witness set as part of their input use a simple, unsofisticated version of the LLL algorithm.

References:

Daniel Bates, Jonathan Hauenstein, Timothy McCoy, Chris Peterson, and Andrew Sommese; Recovering exact results from inexact numerical data in algebraic geometry; Published in Experimental Mathematics 22(1) on pages 38-50 in 2013

Procedures:

D.8.7.1 substAll

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: substAll(v,p); poly v, list p

Return: poly: the polynomial obtained from v by substituting the elements of p for the ring variables

Note: The list p should have as many elements as there are ring variables.

Example:

```
LIB "recover.lib";
ring r=0,(x,y,z),dp;
poly v=x+y+z;
list p=7/11,5/11,-1/11;
poly f=substAll(v,p);
f;
↦ 1
```

D.8.7.2 veronese

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: veronese(d,p); int d, list p

Return: ideal: the image of the point p under the degree d Veronese embedding

Note: The list p should have as many elements as there are ring variables. The order of the points in the returned ideal corresponds to the order of the monomials in maxideal(d).

Example:

```
LIB "recover.lib";
ring R=0,(x,y,z),dp;
list p=2,3,5;
ideal V=veronese(1,p);
V;
↦ V[1]=2
↦ V[2]=3
↦ V[3]=5
V=veronese(2,p);
V;
↦ V[1]=25
↦ V[2]=15
↦ V[3]=9
↦ V[4]=10
↦ V[5]=6
↦ V[6]=4
```

See also: Section 5.1.88 [maxideal], page 219.

D.8.7.3 getRelations

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: getRelations(p,D,C); list p, int D, bigint C

Return: list K: a list of ideals; the ideals contain homogeneous polynomial relations of degree <=D between the components of the point p

Note: This procedure uses only the images of the one point p under the Veronese embeddings to find homogeneous polynomial relations.

Example:

```
LIB "recover.lib";
ring r=(complex,50),(x,y,z),dp;
list p=1,-1,0.5;
getRelations(p,2,10000);
↦ [1]:
```

```
↦    _[1]=x+y
↦    _[2]=x-2*z
↦ [2]:
↦    _[1]=xz+yz
↦    _[2]=xy+y2
↦    _[3]=x2-y2
↦    _[4]=-xy-xz+yz
↦    _[5]=-xz+2*z2
```

See also: Section D.8.7.4 [getRelationsRadical], page 1863.

D.8.7.4 getRelationsRadical

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: getRelationsRadical(P,D,C); list P, int D, bigint C

Return: list K: a list of ideals; the ideals contain homogeneous polynomial relations of degree <=D between the components of the points in P

Note: This procedure uses random linear combination of the Veronese embeddings of all points in P to find homogeneous polynomial relations.

Example:

```
LIB "recover.lib";
ring r=(complex,50),(x,y,z),dp;
list p1=1,-1,0.5;
list p2=1,0,-1;
list P=list(p1)+list(p2);
getRelationsRadical(P,2,10**5);
↦ [1]:
↦    _[1]=-2*x2+2*y2+yz+2*z2
```

See also: Section D.8.7.3 [getRelations], page 1862.

D.8.7.5 gaussRowWithoutPerm

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: gaussRowWithoutPerm(M); M a matrix of constant polynomials

Return: matrix: basic Gaussian row reduction of M, just without permuting the rows

Example:

```
LIB "recover.lib";
ring r=0,x,dp;
matrix M[5][4]=0,0,2,1,4,5,1,3,0,9,2,0,8,1,0,6,0,9,4,1;
print(M);
↦ 0,0,2,1,
↦ 4,5,1,3,
↦ 0,9,2,0,
↦ 8,1,0,6,
↦ 0,9,4,1
print(gaussRowWithoutPerm(M));
↦ 0,0,2,1,
↦ 4,5,1,3,
↦ 0,9,2,0,
↦ 0,0,0,0,
↦ 0,0,0,0
```

D.8.7.6 gaussColWithoutPerm

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: gaussColWithoutPerm(M); M a matrix of constant polynomials

Return: matrix: basic Gaussian column reduction of M, just without permuting the columns

Example:

```
LIB "recover.lib";
ring r=0,x,dp;
matrix M[3][4]=0,1,0,2,1,2,3,4,1,0,5,0;
print(M);
↦ 0,1,0,2,
↦ 1,2,3,4,
↦ 1,0,5,0
print(gaussColWithoutPerm(M));
↦ 0,1,0,0,
↦ 1,2,0,0,
↦ 1,0,2,0
```

D.8.7.7 getWitnessSet

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: getWitnessSet();

Assume: There is a text-document "main_data" in the current directory which was produced by Bertini.
The basefield is the field of real numbers or the field of complex numbers.

Return: list; a list P of lists p_i of numbers: P a set of witness points

Note: Reads the file "main_data", searches the strings containing the witness points, and converts them into floating point numbers.

Example:

```
LIB "recover.lib";
//First, we write the input file for bertini, then run bertini
ring r=0,(x,y,z),dp;
ideal I=(x-y)*(y-z)*(x-z);
writeBertiniInput(I,40);
system("sh","bertini input");
↦
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
```

```
↦ CoeffBound: 6.000000000000e+00, DegreeBound: 3.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 160
↦
↦
↦ Tracking regeneration codim 1 of 1: 3 paths to track.
↦ Tracking path 0 of 3
↦ Tracking path 1 of 3
↦ Tracking path 2 of 3
↦
↦ Sorting codimension 1 of 1: 3 paths to sort.
↦ Sorting 0 of 3
↦ Sorting 1 of 3
↦ Sorting 2 of 3
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ---------------------------------------------------------------------------------
↦ | 1   |   3       |   3            |  3          |  0       |  0         \
  |   0           |  0
↦ ---------------------------------------------------------------------------------
↦ |total|   3
↦
↦ ****************************************************
↦
↦
↦
↦ *************** Witness Set Summary ****************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 1   |   3            |  3          |  0
↦ -------------------------------------------------
↦
↦ ****************************************************
↦
↦
↦ Calculating traces for codimension 1.
↦ Calculating 0 of 3
↦ Calculating 1 of 3
↦ Calculating 2 of 3
↦
↦ Using combinatorial trace test to decompose codimension 1.
↦
↦
```

```
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   1       |   0        |   0        |  3
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ *****************************************************
↦
↦ 0
//Then we change the ring and extract the witness set from main_data
ring R=(complex,40,i),(x,y,z),dp;
list P=getWitnessSet();
P;
↦ [1]:
↦    [1]:
↦ (0.2301252864658052820651665411643051571221-i*0.1918430853241043201972832\
   935803665200436)
↦    [2]:
↦ (0.3093122929262355666347230675902153443977-i*0.2581686392628814488538693\
   878647114614651)
↦    [3]:
↦ (0.2301252864658052820651665411643051571221-i*0.1918430853241043201972832\
   935803665200436)
↦ [2]:
↦    [1]:
↦ (0.2671478460238462313839021275681175756573-i*0.2437218849893980178650402\
   101091730112661)
↦    [2]:
↦ (0.2671478460238462313839021275681175756573-i*0.2437218849893980178650402\
   101091730112661)
↦    [3]:
↦ (0.2080234853228090863025992820595946767208-i*0.1456494783567712591063986\
   754901131676131)
↦ [3]:
↦    [1]:
↦ (0.07513685470326444549729102379894746071941-i*0.2737502137489732099962287\
   7349485825946751)
↦    [2]:
↦ (0.3641474435957191379047964747452006894328-i*0.1485254697157806682678202\
   318014107599763)
↦    [3]:
↦ (0.3641474435957191379047964747452006894328-i*0.1485254697157806682678202\
   318014107599763)
```

D.8.7.8 writeBertiniInput

Procedure from library recover.lib (see Section D.8.7 [recover_lib], page 1861).

Usage: writeBertiniInput(J); ideal J

Return: none; writes the input-file for bertini using the polynomials given by J as functions

Note: Either creates a file named input in the current directory or overwrites the existing one.
If you want to pass different parameters to bertini, you can edit the produced input file or redefine this procedure.

Example:

```
LIB "recover.lib";
ring r=0,(x,y,z),dp;
poly f1=x+y+z;
poly f2=x2+xy+y2;
ideal I=f1,f2;
writeBertiniInput(I,300);
```

D.8.7.9 num_prime_decom

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_prime_decom(I,D); ideal I, int D
D a bound to the degree of the elements of the components of a prime decomposition of I.

Return: list of ideals: each of the ideals a prime component of the radical of I

Remarks: Uses Bertini.

Note: Should only be called from a ring over the rational numbers.

Example:

```
LIB "recover.lib";
ring R=0,(x,y,z),dp;
ideal I=(x+y)*(y+2z), (x+y)*(x-3z);
int D=2;
int Prec=300;
num_prime_decom(I,D,Prec);
↦
↦     Bertini(TM) v1.6
↦      (May 22, 2018)
↦
↦   D.J. Bates, J.D. Hauenstein,
↦   A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 8.000000000000e+00, DegreeBound: 2.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 1024
↦
↦
↦ Tracking regeneration codim 1 of 2: 2 paths to track.
↦ Tracking path 0 of 2
↦ Tracking path 1 of 2
```

```
↦
↦ Sorting codimension 1 of 2: 2 paths to sort.
↦ Sorting 0 of 2
↦ Sorting 1 of 2
↦
↦ Preparing regeneration codim 2 of 2: 1 witness point to move.
↦ Moving 0 of 1
↦
↦ Tracking regeneration codim 2 of 2: 2 paths to track.
↦ Tracking path 0 of 2
↦ Tracking path 1 of 2
↦
↦ Sorting codimension 2 of 2: 2 paths to sort.
↦ Sorting 0 of 2
↦ Sorting 1 of 2
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ -----------------------------------------------------------------------------------
↦ | 1   |   2       |   1            |  1          |  0       |  1         \
  |   0           |  0
↦ | 2   |   2       |   2            |  1          |  1       |  0         \
  |   0           |  0
↦ -----------------------------------------------------------------------------------
↦ |total|   4
↦
↦ ****************************************************
↦
↦
↦ Removing junk points from codimension 2: 1 endpoints to check.
↦ Checking 0 of 1
↦
↦
↦ *************** Witness Set Summary ****************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 1   |   1            |  1          |  0
↦ | 2   |   1            |  1          |  0
↦ -------------------------------------------------
↦
↦ ****************************************************
↦
↦
```

```
↦ Calculating traces for codimension 1.
↦ Calculating 0 of 1
↦ 
↦ Using combinatorial trace test to decompose codimension 1.
↦ 
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 1
↦ 
↦ 
↦ ************* Witness Set Decomposition *************
↦ 
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   1       |   0        |   0        |  1
↦ |   0       |   1        |   1        |  0
↦ -----------------------------------------------------
↦ 
↦ ************** Decomposition by Degree **************
↦ 
↦ Dimension 0: 1 classified component
↦ -----------------------------------------------------
↦    degree 1: 1 component
↦ 
↦ *****************************************************
↦ 
↦ 0
↦ empty list
//Let us compare that to the result of primdecSY:
primdecSY(I);
↦ [1]:
↦    [1]:
↦       _[1]=x+y
↦    [2]:
↦       _[1]=x+y
↦ [2]:
↦    [1]:
↦       _[1]=y+2z
↦       _[2]=x-3z
↦    [2]:
↦       _[1]=y+2z
↦       _[2]=x-3z
```

D.8.7.10 num_prime_decom1

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_prime_decom1(P,D,C); list P, int D, bigint C
P a list of lists representing a witness point set representing an ideal I D should be a bound to the degree of the elements of the components of the prime decomposition of I
C the number with which the images of the Veronese embeddings are multiplied

Return: list of ideals: each of the ideals a prime component of the radical of I

Note: Should only be called from a ring over the complex numbers.

Example:

```
LIB "recover.lib";
//First, we compute a prime decomposition of the ideal I=x+y;
ring R1=(complex,300,IUnit),(x,y),dp;
list p1=1,-1;
list P=list(p1);
int D=2;
bigint C=bigint(10)**300;
num_prime_decom1(P,D,C);
↦ [1]:
↦    _[1]=x+y
//Now, we try to obtain a prime decomposition of the ideal I=(x+y)*(y+2z), (x+y)*(x-3
ring R2=(complex,20,IUnit),(x,y,z),dp;
p1=1.7381623928,-1.7381623928,0.2819238763;
list p2=-3.578512854,2.385675236,-1.192837618;
P=p1,p2;
num_prime_decom1(P,D,10000);
↦ [1]:
↦    _[1]=x+y
↦ [2]:
↦    _[1]=-2*x-3*y
↦    _[2]=-y^2+x*z+z^2
↦    _[3]=x*z+y*z-z^2
//Now, we look at the result of a purely symbolic algorithm
ring r2=0,(x,y,z),dp;
ideal I=(x+y)*(y+2z), (x+y)*(x-3z);
primdecSY(I);
↦ [1]:
↦    [1]:
↦       _[1]=x+y
↦    [2]:
↦       _[1]=x+y
↦ [2]:
↦    [1]:
↦       _[1]=y+2z
↦       _[2]=x-3z
↦    [2]:
↦       _[1]=y+2z
↦       _[2]=x-3z
//If you compare the results, you may find that they don't match.
//Most likely, the hybrid algorithm got the second component wrong. This is due to th
//way the algorithm looks for homogeneous polynomial relations, and the specific vers
//of the LLL algorithm used here (an implementation into Singular of a rather simple
//version which allows real input). It looks in degree 1, finds one relation and is
//thereafter unable to see a second one. Then it moves on to degree 2 and finds
//relations containing degree-1 relations as a factor.
```

D.8.7.11 num_radical_via_decom

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_radical_via_decom(I,D); ideal I, int D
D a bound to the degree of the elements of the components.

Return: ideal: the radical of I

Remarks: Uses Bertini.
This procedure merely calls num_prime_decom with the same input and then intersects the returned components.

Note: Should only be called from a ring over the rational numbers.

Example:

```
LIB "recover.lib";
//First, we attempt to compute the radical via the hybrid algorithm.
ring R=0,(x,y,z),dp;
ideal I=(x+y)^2*(y+2z)^3, (x+y)^3*(x-3z)^2;
int D=2;
int Prec=300;
ideal numRad=num_radical_via_decom(I,D,Prec);
↦
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 1.280000000000e+02, DegreeBound: 5.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 1024
↦
↦
↦ Tracking regeneration codim 1 of 2: 5 paths to track.
↦ Tracking path 0 of 5
↦ Tracking path 1 of 5
↦ Tracking path 2 of 5
↦ Tracking path 3 of 5
↦ Tracking path 4 of 5
↦
↦ Sorting codimension 1 of 2: 5 paths to sort.
↦ Sorting 0 of 5
↦ Sorting 1 of 5
↦ Sorting 2 of 5
↦ Sorting 3 of 5
↦ Sorting 4 of 5
↦
↦ Preparing regeneration codim 2 of 2: 12 witness points to move.
↦ Moving 0 of 12
↦ Moving 1 of 12
↦ Moving 2 of 12
↦ Moving 3 of 12
↦ Moving 4 of 12
↦ Moving 5 of 12
↦ Moving 6 of 12
↦ Moving 7 of 12
↦ Moving 8 of 12
↦ Moving 9 of 12
↦ Moving 10 of 12
↦ Moving 11 of 12
↦
↦ Tracking regeneration codim 2 of 2: 15 paths to track.
↦ Tracking path 0 of 15
↦ Tracking path 1 of 15
↦ Tracking path 2 of 15
↦ Tracking path 3 of 15
↦ Tracking path 4 of 15
↦ Tracking path 5 of 15
↦ Tracking path 6 of 15
↦ Tracking path 7 of 15
↦ Tracking path 8 of 15
↦ Tracking path 9 of 15
↦ Tracking path 10 of 15
↦ Tracking path 11 of 15
↦ Tracking path 12 of 15
↦ Tracking path 13 of 15
↦ Tracking path 14 of 15
↦
↦ Sorting codimension 2 of 2: 15 paths to sort.
↦ Sorting 0 of 15
↦ Sorting 1 of 15
↦ Sorting 2 of 15
↦ Sorting 3 of 15
↦ Sorting 4 of 15
↦ Sorting 5 of 15
↦ Sorting 6 of 15
↦ Sorting 7 of 15
↦ Sorting 8 of 15
↦ Sorting 9 of 15
↦ Sorting 10 of 15
↦ Sorting 11 of 15
↦ Sorting 12 of 15
↦ Sorting 13 of 15
↦ Sorting 14 of 15
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ---------------------------------------------------------------------------------
↦ | 1   |   5       |   2            |  0          |  2       |  3        \
  |   0           |  0
```

```
↦ | 2   |   15      |   15           | 0          | 15      | 0         \
  |   0           | 0
↦ ------------------------------------------------------------------------------
↦ |total|   20
↦
↦ ****************************************************
↦
↦
↦ Removing junk points from codimension 2: 2 endpoints to check.
↦ Checking 0 of 2
↦ Checking 1 of 2
↦
↦
↦ *************** Witness Set Summary ****************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 1   |   1            |  0          |  1
↦ | 2   |   1            |  0          |  1
↦ -------------------------------------------------
↦
↦ ****************************************************
↦
↦
↦ Calculating traces for codimension 1.
↦ Calculating 0 of 1
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 1
↦
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   1       |   1        |   1        |  0
↦ |   0       |   1        |   1        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 1: 1 classified component
↦ -----------------------------------------------------
↦    degree 1: 1 component
↦
↦ Dimension 0: 1 classified component
↦ -----------------------------------------------------
↦    degree 1: 1 component
↦
↦ *****************************************************
```

```
↦
↦ Witness Points Deflated: 2
↦ 0
numRad;
↦ numRad[1]=1
//Then we compute the radical symbolically and compare the results.
ideal Rad=radical(I);
Rad;
↦ Rad[1]=xy+y2+2xz+2yz
↦ Rad[2]=x2-y2-5xz-5yz
reduce(Rad,std(numRad));
↦ _[1]=0
↦ _[2]=0
reduce(numRad,std(Rad));
↦ _[1]=1
```

See also: Section D.8.7.9 [num_prime_decom], page 1867; Section D.8.7.12 [num_radical_via_randlincom], page 1874.

D.8.7.12 num_radical_via_randlincom

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_radical_via_randlincom(I,D); ideal I, int D
D a bound to the degree of the elements of the components.

Return: ideal: the radical of I

Remarks: Uses Bertini.
Instead of using the images of the Veronese embeddings of each individual witness point, this procedure first computes a random linear combination of those images and searches for homogeneous polynomial relations for this linear combination.

Note: Should only be called from a ring over the rational numbers.

Example:

```
LIB "recover.lib";
//First, we attempt to compute the radical via the hybrid algorithm.
ring R=0,(x,y,z),dp;
ideal I=(x+y)^2*(y+2z)^3, (x+y)^3*(x-3z)^2;
int D=2;
int Prec=300;
ideal numRad=num_radical_via_randlincom(I,D,Prec);
↦
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
  that you have
```

```
↦ setup the following tolerances appropriately:
↦ CoeffBound: 1.280000000000e+02, DegreeBound: 5.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 1024
↦
↦
↦ Tracking regeneration codim 1 of 2: 5 paths to track.
↦ Tracking path 0 of 5
↦ Tracking path 1 of 5
↦ Tracking path 2 of 5
↦ Tracking path 3 of 5
↦ Tracking path 4 of 5
↦
↦ Sorting codimension 1 of 2: 5 paths to sort.
↦ Sorting 0 of 5
↦ Sorting 1 of 5
↦ Sorting 2 of 5
↦ Sorting 3 of 5
↦ Sorting 4 of 5
↦
↦ Preparing regeneration codim 2 of 2: 12 witness points to move.
↦ Moving 0 of 12
↦ Moving 1 of 12
↦ Moving 2 of 12
↦ Moving 3 of 12
↦ Moving 4 of 12
↦ Moving 5 of 12
↦ Moving 6 of 12
↦ Moving 7 of 12
↦ Moving 8 of 12
↦ Moving 9 of 12
↦ Moving 10 of 12
↦ Moving 11 of 12
↦
↦ Tracking regeneration codim 2 of 2: 15 paths to track.
↦ Tracking path 0 of 15
↦ Tracking path 1 of 15
↦ Tracking path 2 of 15
↦ Tracking path 3 of 15
↦ Tracking path 4 of 15
↦ Tracking path 5 of 15
↦ Tracking path 6 of 15
↦ Tracking path 7 of 15
↦ Tracking path 8 of 15
↦ Tracking path 9 of 15
↦ Tracking path 10 of 15
↦ Tracking path 11 of 15
↦ Tracking path 12 of 15
↦ Tracking path 13 of 15
↦ Tracking path 14 of 15
↦
↦ Sorting codimension 2 of 2: 15 paths to sort.
↦ Sorting 0 of 15
↦ Sorting 1 of 15
↦ Sorting 2 of 15
↦ Sorting 3 of 15
↦ Sorting 4 of 15
↦ Sorting 5 of 15
↦ Sorting 6 of 15
↦ Sorting 7 of 15
↦ Sorting 8 of 15
↦ Sorting 9 of 15
↦ Sorting 10 of 15
↦ Sorting 11 of 15
↦ Sorting 12 of 15
↦ Sorting 13 of 15
↦ Sorting 14 of 15
↦ 
↦ 
↦ ************ Regenerative Cascade Summary ************
↦ 
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦ 
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ---------------------------------------------------------------------------------
↦ | 1   |   5       |   2            | 0          |  2       |  3         \
  |   0           |  0
↦ | 2   |   15      |   15           | 0          |  15      |  0         \
  |   0           |  0
↦ ---------------------------------------------------------------------------------
↦ |total|   20
↦ 
↦ ****************************************************
↦ 
↦ 
↦ Removing junk points from codimension 2: 2 endpoints to check.
↦ Checking 0 of 2
↦ Checking 1 of 2
↦ 
↦ 
↦ *************** Witness Set Summary ***************
↦ 
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦ 
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 1   |   1            |  0          |  1
↦ | 2   |   1            |  0          |  1
↦ -------------------------------------------------
↦ 
↦ ****************************************************
↦ 
↦ 
↦ Calculating traces for codimension 1.
↦ Calculating 0 of 1
↦
↦ Using combinatorial trace test to decompose codimension 1.
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 1
↦
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   1       |   0        |   0        |  1
↦ |   0       |   1        |   1        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 0: 1 classified component
↦ -----------------------------------------------------
↦    degree 1: 1 component
↦
↦ *****************************************************
↦
↦ Witness Points Deflated: 2
↦ 0
numRad;
↦ numRad[1]=0
//Then we compute the radical symbolically and compare the results.
ideal Rad=radical(I);
Rad;
↦ Rad[1]=xy+y2+2xz+2yz
↦ Rad[2]=x2-y2-5xz-5yz
reduce(Rad,std(numRad));
↦ _[1]=xy+y2+2xz+2yz
↦ _[2]=x2-y2-5xz-5yz
reduce(numRad,std(Rad));
↦ _[1]=0
```

See also: Section D.8.7.11 [num_radical_via_decom], page 1870.

D.8.7.13 num_radical1

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_radical1(P,D,C); list P, int D, bigint C
P a list of lists representing a witness point set representing an ideal I D should be a bound to the degree of the elements of the components C the number with which the images of the Veronese embeddings are multiplied

Return: list of ideals: each of the ideals a prime component of the radical of I

Remarks: This procedure merely calls num_prime_decom1 with the same input and then intersects the returned components.

Note: Should only be called from a ring over the complex numbers.

Example:

```
LIB "recover.lib";
//First, we write the input file for bertini and compute the radical symbolically.
ring r=0,(x,y,z),dp;
ideal I=4xy2-4z3,-2x2y+5xz2;
ideal Rad=radical(I);
writeBertiniInput(I,100);
//Then we attempt to compute the radical via the hybrid algorithm.
ring R=(complex,100,i),(x,y,z),dp;
system("sh","bertini input");
↦
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 8.000000000000e+00, DegreeBound: 3.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 352
↦
↦
↦ Tracking regeneration codim 1 of 2: 3 paths to track.
↦ Tracking path 0 of 3
↦ Tracking path 1 of 3
↦ Tracking path 2 of 3
↦
↦ Sorting codimension 1 of 2: 3 paths to sort.
↦ Sorting 0 of 3
↦ Sorting 1 of 3
↦ Sorting 2 of 3
↦
↦ Preparing regeneration codim 2 of 2: 6 witness points to move.
↦ Moving 0 of 6
↦ Moving 1 of 6
↦ Moving 2 of 6
↦ Moving 3 of 6
↦ Moving 4 of 6
↦ Moving 5 of 6
↦
↦ Tracking regeneration codim 2 of 2: 9 paths to track.
↦ Tracking path 0 of 9
↦ Tracking path 1 of 9
↦ Tracking path 2 of 9
↦ Tracking path 3 of 9
↦ Tracking path 4 of 9
↦ Tracking path 5 of 9
↦ Tracking path 6 of 9
↦ Tracking path 7 of 9
↦ Tracking path 8 of 9
↦
↦ Sorting codimension 2 of 2: 9 paths to sort.
↦ Sorting 0 of 9
↦ Sorting 1 of 9
↦ Sorting 2 of 9
↦ Sorting 3 of 9
↦ Sorting 4 of 9
↦ Sorting 5 of 9
↦ Sorting 6 of 9
↦ Sorting 7 of 9
↦ Sorting 8 of 9
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ----------------------------------------------------------------------------------
↦ | 1   |   3       |   0            |  0          |  0       |  3         \
  |   0           |  0
↦ | 2   |   9       |   9            |  1          |  8       |  0         \
  |   0           |  0
↦ ----------------------------------------------------------------------------------
↦ |total|   12
↦
↦ ****************************************************
↦
↦
↦
↦ *************** Witness Set Summary ***************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 2   |   3            |  1          |  2
↦ -------------------------------------------------
↦
↦ ****************************************************
↦
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 3
↦ Calculating 1 of 3
↦ Calculating 2 of 3
↦
```

```
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   0       |   3        |   3        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 0: 3 classified components
↦ -----------------------------------------------------
↦    degree 1: 3 components
↦
↦ *****************************************************
↦
↦ Witness Points Deflated: 2
↦ 0
list P=getWitnessSet();
int D=2;
bigint C=bigint(10)**30;
ideal Rad1=num_radical1(P,D,C);
//Lastly, we compare the results.
Rad1;
↦ Rad1[1]=yz-0.4*z2
↦ Rad1[2]=xz-6.25*z2
↦ Rad1[3]=xy-2.5*z2
ideal Rad=fetch(r,Rad);
Rad;
↦ Rad[1]=5*yz-2*z2
↦ Rad[2]=4*xz-25*z2
↦ Rad[3]=2*xy-5*z2
reduce(Rad,std(Rad1));
↦ // ** groebner base computations with inexact coefficients can not be tru\
  sted due to rounding errors
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
reduce(Rad1,std(Rad));
↦ // ** groebner base computations with inexact coefficients can not be tru\
  sted due to rounding errors
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
```

See also: Section D.8.7.10 [num_prime_decom1], page 1869; Section D.8.7.14 [num_radical2], page 1880.

D.8.7.14 num_radical2

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_radical2(P,D,C); list P, int D, bigint C
P a list of lists representing a witness point set representing an ideal I D should be a

bound to the degree of the elements of the components C the number with which the images of the Veronese embeddings are multiplied

Return: list of ideals: each of the ideals a prime component of the radical of I

Remarks: Instead of using the images of the Veronese embeddings of each individual witness point, this procedure first computes a random linear combination of those images and searches for homogeneous polynomial relations for this linear combination.

Note: Should only be called from a ring over the complex numbers.

Example:

```
LIB "recover.lib";
//First, we write the input file for bertini and compute the radical symbolically.
ring r=0,(x,y,z),dp;
ideal I=4xy2-4z3,-2x2y+5xz2;
ideal Rad=radical(I);
writeBertiniInput(I,100);
//Then we attempt to compute the radical via the hybrid algorithm.
ring R=(complex,100,i),(x,y,z),dp;
system("sh","bertini input");
↦ 
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦ 
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦ 
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦ 
↦ 
↦ 
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 8.000000000000e+00, DegreeBound: 3.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 352
↦ 
↦ 
↦ Tracking regeneration codim 1 of 2: 3 paths to track.
↦ Tracking path 0 of 3
↦ Tracking path 1 of 3
↦ Tracking path 2 of 3
↦ 
↦ Sorting codimension 1 of 2: 3 paths to sort.
↦ Sorting 0 of 3
↦ Sorting 1 of 3
↦ Sorting 2 of 3
↦ 
↦ Preparing regeneration codim 2 of 2: 6 witness points to move.
↦ Moving 0 of 6
↦ Moving 1 of 6
↦ Moving 2 of 6
↦ Moving 3 of 6
↦ Moving 4 of 6
↦ Moving 5 of 6
↦
↦ Tracking regeneration codim 2 of 2: 9 paths to track.
↦ Tracking path 0 of 9
↦ Tracking path 1 of 9
↦ Tracking path 2 of 9
↦ Tracking path 3 of 9
↦ Tracking path 4 of 9
↦ Tracking path 5 of 9
↦ Tracking path 6 of 9
↦ Tracking path 7 of 9
↦ Tracking path 8 of 9
↦
↦ Sorting codimension 2 of 2: 9 paths to sort.
↦ Sorting 0 of 9
↦ Sorting 1 of 9
↦ Sorting 2 of 9
↦ Sorting 3 of 9
↦ Sorting 4 of 9
↦ Sorting 5 of 9
↦ Sorting 6 of 9
↦ Sorting 7 of 9
↦ Sorting 8 of 9
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ---------------------------------------------------------------------------------
↦ | 1   |   3       |   0            |  0          |  0       |  3         \
  |   0           |  0
↦ | 2   |   9       |   9            |  1          |  8       |  0         \
  |   0           |  0
↦ ---------------------------------------------------------------------------------
↦ |total|   12
↦
↦ ****************************************************
↦
↦
↦
↦ *************** Witness Set Summary ****************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 2   |   3            |  1          |  2
↦ -------------------------------------------------
```

```
↦
↦ ****************************************************
↦
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 3
↦ Calculating 1 of 3
↦ Calculating 2 of 3
↦
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   0       |   3        |   3        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 0: 3 classified components
↦ -----------------------------------------------------
↦    degree 1: 3 components
↦
↦ *****************************************************
↦
↦ Witness Points Deflated: 2
↦ 0
list P=getWitnessSet();
int D=2;
bigint C=bigint(10)**30;
ideal Rad2=num_radical2(P,D,C);
↦ [1]:
↦    _[1]=5*yz-2*z2
↦    _[2]=-8*xy+4*xz-5*z2
//Lastly, we compare the results.
Rad2;
↦ Rad2[1]=5*yz-2*z2
↦ Rad2[2]=-8*xy+4*xz-5*z2
ideal Rad=fetch(r,Rad);
Rad;
↦ Rad[1]=5*yz-2*z2
↦ Rad[2]=4*xz-25*z2
↦ Rad[3]=2*xy-5*z2
reduce(Rad,std(Rad2));
↦ // ** groebner base computations with inexact coefficients can not be tru\
   sted due to rounding errors
↦ _[1]=0
↦ _[2]=4*xz-25*z2
↦ _[3]=xz-6.25*z2
reduce(Rad2,std(Rad));
↦ // ** groebner base computations with inexact coefficients can not be tru\
   sted due to rounding errors
↦ _[1]=0
↦ _[2]=0
```

See also: Section D.8.7.13 [num_radical1], page 1877.

D.8.7.15 num_elim

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_elim(I,f,D); ideal I, poly f, int D
f the product of the ring variables you want to eliminate D a bound to the degree of the elements of the components

Return: ideal: the ideal obtained from I by eliminating the variables specified in f

Remarks: This procedure uses Bertini to compute a set of witness points for I, projects them onto the components corresponding to the variables specified in f and then proceeds as num_radical_via_randlincom.

Note: Should only be called from a ring over the rational numbers.

Example:

```
LIB "recover.lib";
ring r=0,(x,y,z),dp;
poly f1=x-y;
poly f2=z*(x+3y);
poly f3=z*(x2+y2);
ideal I=f1,f2,f3;
//First, we attempt to compute the elimination ideal with the hybrid algorithm.
ideal E1=num_elim(I,z,3,200);
↦
↦    Bertini(TM) v1.6
↦     (May 22, 2018)
↦
↦  D.J. Bates, J.D. Hauenstein,
↦  A.J. Sommese, C.W. Wampler
↦
↦ (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 4.000000000000e+00, DegreeBound: 3.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 672
↦
↦
↦ Tracking regeneration codim 1 of 2: 3 paths to track.
↦ Tracking path 0 of 3
↦ Tracking path 1 of 3
↦ Tracking path 2 of 3
↦
↦ Sorting codimension 1 of 2: 3 paths to sort.
↦ Sorting 0 of 3
↦ Sorting 1 of 3
↦ Sorting 2 of 3
```

```
↦
↦ Preparing regeneration codim 2 of 2: 3 witness points to move.
↦ Moving 0 of 3
↦ Moving 1 of 3
↦ Moving 2 of 3
↦
↦ Tracking regeneration codim 2 of 2: 6 paths to track.
↦ Tracking path 0 of 6
↦ Tracking path 1 of 6
↦ Tracking path 2 of 6
↦ Tracking path 3 of 6
↦ Tracking path 4 of 6
↦ Tracking path 5 of 6
↦
↦ Sorting codimension 2 of 2: 6 paths to sort.
↦ Sorting 0 of 6
↦ Sorting 1 of 6
↦ Sorting 2 of 6
↦ Sorting 3 of 6
↦ Sorting 4 of 6
↦ Sorting 5 of 6
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ---------------------------------------------------------------------------------
↦ | 1   |   3       |   0            |  0          |  0       |  3         \
  |   0           |  0
↦ | 2   |   6       |   2            |  2          |  0       |  3         \
  |   1           |  0
↦ ---------------------------------------------------------------------------------
↦ |total|   9
↦
↦ ****************************************************
↦
↦
↦
↦ *************** Witness Set Summary ***************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 2   |   2            |  2          |  0
↦ -------------------------------------------------
↦
↦ ****************************************************
```

```
↦
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 2
↦ Calculating 1 of 2
↦
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   0       |   2        |   2        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 0: 2 classified components
↦ -----------------------------------------------------
↦    degree 1: 2 components
↦
↦ *****************************************************
↦
↦ 0
//Now, we compute the elimination ideal symbolically.
ideal E2=elim(I,z);
//Lastly, we compare the results.
E1;
↦ E1[1]=0
E2;
↦ E2[1]=x-y
```

D.8.7.16 num_elim1

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: num_elim1(P,D,C,v); list P, int D, bigint C, intvec v
P a list of lists representing a witness point set representing an ideal J D should be a bound to the degree of the elements of the components C the number with which the images of the Veronese embeddings are multiplied v an intvec specifying the numbers/positions of the variables to be eliminated

Return: ideal: the ideal obtained from J by eliminating the variables specified in v

Remarks: This procedure just canonically projects the witness points onto the components specified in the intvec v and then applies num_radical1 to the resulting points.

Note: Should only be called from a ring over the complex numbers.

Example:

```
LIB "recover.lib";
//First, we write the input file for bertini and compute the elimination ideal
//symbolically.
ring r=0,(x,y,z),dp;
poly f1=x-y;
poly f2=z*(x+3y);
poly f3=z*(x2+y2);
ideal J=f1,f2,f3;
ideal E2=elim(J,z);
writeBertiniInput(J,100);
//Then we attempt to compute the elimination ideal via the hybrid algorithm.
ring R=(complex,100,i),(x,y,z),dp;
system("sh","bertini input");
↦
↦     Bertini(TM) v1.6
↦      (May 22, 2018)
↦
↦   D.J. Bates, J.D. Hauenstein,
↦   A.J. Sommese, C.W. Wampler
↦
↦  (using GMP v6.0.0, MPFR v3.1.2)
↦
↦
↦
↦ NOTE: You have requested to use adaptive path tracking.  Please make sure\
   that you have
↦ setup the following tolerances appropriately:
↦ CoeffBound: 4.000000000000e+00, DegreeBound: 3.000000000000e+00
↦ AMPSafetyDigits1: 1, AMPSafetyDigits2: 1, AMPMaxPrec: 352
↦
↦
↦ Tracking regeneration codim 1 of 2: 3 paths to track.
↦ Tracking path 0 of 3
↦ Tracking path 1 of 3
↦ Tracking path 2 of 3
↦
↦ Sorting codimension 1 of 2: 3 paths to sort.
↦ Sorting 0 of 3
↦ Sorting 1 of 3
↦ Sorting 2 of 3
↦
↦ Preparing regeneration codim 2 of 2: 3 witness points to move.
↦ Moving 0 of 3
↦ Moving 1 of 3
↦ Moving 2 of 3
↦
↦ Tracking regeneration codim 2 of 2: 6 paths to track.
↦ Tracking path 0 of 6
↦ Tracking path 1 of 6
↦ Tracking path 2 of 6
↦ Tracking path 3 of 6
↦ Tracking path 4 of 6
↦ Tracking path 5 of 6
↦
↦ Sorting codimension 2 of 2: 6 paths to sort.
↦ Sorting 0 of 6
↦ Sorting 1 of 6
↦ Sorting 2 of 6
↦ Sorting 3 of 6
↦ Sorting 4 of 6
↦ Sorting 5 of 6
↦
↦
↦ ************ Regenerative Cascade Summary ************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim|   paths   |witness superset| nonsingular | singular |nonsolutions\
  | inf endpoints | other bad endpoints
↦ ----------------------------------------------------------------------------------
↦ | 1   |   3       |   0            |  0          |  0       |  3         \
  |   0           |  0
↦ | 2   |   6       |   2            |  2          |  0       |  3         \
  |   1           |  0
↦ ----------------------------------------------------------------------------------
↦ |total|   9
↦
↦ ****************************************************
↦
↦
↦
↦ *************** Witness Set Summary ****************
↦
↦ NOTE: nonsingular vs singular is based on rank deficiency and identical e\
  ndpoints
↦
↦ |codim| witness points | nonsingular | singular
↦ -------------------------------------------------
↦ | 2   |   2            |  2          |  0
↦ -------------------------------------------------
↦
↦ ****************************************************
↦
↦
↦ Calculating traces for codimension 2.
↦ Calculating 0 of 2
↦ Calculating 1 of 2
↦
↦
↦ ************* Witness Set Decomposition *************
↦
↦ | dimension | components | classified | unclassified
↦ -----------------------------------------------------
↦ |   0       |   2        |   2        |  0
↦ -----------------------------------------------------
↦
↦ ************** Decomposition by Degree **************
↦
↦ Dimension 0: 2 classified components
↦ -----------------------------------------------------
↦    degree 1: 2 components
```

```
↦
↦ ****************************************************
↦
↦ 0
list P=getWitnessSet();
intvec v=3;
bigint C=bigint(10)**25;
ideal E1=num_elim1(P,2,C,v);
//Lastly, we compare the results.
E1;
↦ E1[1]=x-y
setring r;
E2;
↦ E2[1]=x-y
```

D.8.7.17 realLLL

Procedure from library `recover.lib` (see Section D.8.7 [recover_lib], page 1861).

Usage: realLLL(M); matrix M

Assume: The columns of M represent a basis of a lattice.
The groundfield is the field of real number or the field of complex numbers, the elements of M are real numbers.

Return: matrix: the columns representing an LLL-reduced basis of the lattice given by M

Example:

```
LIB "recover.lib";
ring r=(real,50),x,dp;
matrix M[5][4]=
1,0,0,0,
0,1,0,0,
0,0,1,0,
0,0,0,1,
5*81726716.91827716, 817267.1691827716, poly(10)**30, 13*81726716.91827716;
matrix L=realLLL(M);
print(L);
↦ 1,   0,                -3,                  -315358105194840436449,
↦ -500,1,                0,                   -630716210389680833,
↦ 0,   0,                0,                   1,
↦ 0,   0,                1,                   -819931073506585134768,
↦ 0,   817267.1691827716,-163453433.83655432,771738001439.2007536172
```

D.8.8 rootisolation_lib

Library: rootisolation.lib

Purpose: implements an algorithm for real root isolation using interval arithmetic

Authors: Dominik Bendle (bendle@rhrk.uni-kl.de)
Janko Boehm (boehm@mathematik.uni-lk.de), supervisor Fachpraktikum
Clara Petroll (petroll@rhrk.uni-kl.de)

Overview: In this library the interval arithmetic from `interval.so` is used. The new type `ivmat`, a matrix consisting of intervals, is implemented as `newstruct`. There are various functions for computations with interval matrices implemented, such as Gaussian elimination for interval matrices.

Interval arithmetic, the interval Newton Step and exclusion methods are used to implement the procedure `rootIsolation`, an algorithm which finds boxes containing elements of the vanishing locus of an ideal. This algorithm is specialised for zero-dimensional radical ideals. The theory about the interval Newton Step is detailed in [2].

Note that interval arithmetic and the aforementioned procedures are intended for rational or real polynomial rings.

References:

[1] Cloud, Kearfott, Moore: Introduction to Interval Analysis, Society for Industrial and Applied Mathematics, 2009
[2] Eisenbud, Grayson, Herzog, Stillman, Vasconcelos: Computational Methods in Commutative Algebra and Algebraic Geometry, Springer Verlag Berlin-Heidelberg, 3. edition 2004
[3] Andrew J. Sommese and Charles W. Wampler: The Numerical Solution of Systems of Polynomials - Arising in Engineering and Science, World Scientific Publishing Co. Pte. Ltd., 2005

Overloads:

[ivmatGet indexing
print ivmatPrint printing
nrows ivmatNrows number of rows
ncols ivmatNcols number of columns
* ivmatMultiplyGeneral matrix multiplication

Procedures: See also: Section D.8.2 [interval_lib], page 1824.

D.8.8.1 bounds

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `bounds(a); a number;`
`bounds(a, b); a, b number;`

Return: interval: if `size(#)==0` it returns the interval `[a, a]`, else the interval `[a,#[1]]`

Example:

```
LIB "rootisolation.lib";
ring R = 0,x,dp;
interval I = bounds(1);
I;
↦ [1, 1]
interval J = bounds(2/3,3);
J;
↦ [2/3, 3]
```

D.8.8.2 length

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `length(I)`; `I interval`

Return: number, the Euclidean length of the interval

Example:

```
LIB "rootisolation.lib";
ring R = 0,x,dp;
interval I = -1,3;
length(I);
↦ 4
I = 1/5,1/3;
length(I);
↦ 2/15
```

D.8.8.3 boxSet

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `boxSet(B, i, I)`; `B box, i int, I interval`

Return: new box `C` where `C[i]==I`

Purpose: modifies a single entry of a box

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y,z),dp;
box B; B;
↦ [0, 0] x [0, 0] x [0, 0]
B = boxSet(B, 2, bounds(-1,1)); B;
↦ [0, 0] x [-1, 1] x [0, 0]
B = boxSet(B, 1, B[2]); B;
↦ [-1, 1] x [-1, 1] x [0, 0]
```

D.8.8.4 ivmatInit

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `ivmatInit(m, n)`; `m, n int`

Return: `mxn` matrix of [0,0]-intervals

Purpose: initialises an interval matrix with [0,0] intervals to ensure the proper structure of the `ivmat` type

Example:

```
LIB "rootisolation.lib";
ring R = 0,x(1..5),dp;
ivmat A = ivmatInit(4, 5); A;
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦ [0, 0],[0, 0],[0, 0],[0, 0],[0, 0]
↦
```

D.8.8.5 ivmatSet

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `ivmatSet(A, i, j, I)`; `A ivmat, i, j, int, I interval`

Return: interval matrix `A` where `A[i][j] == I`

Purpose: modify a single entry of an `ivmat`

Example:

```
LIB "rootisolation.lib";
ring R = 0,x,dp;
ivmat A = ivmatInit(2,2);             A;
↦ [0, 0],[0, 0]
↦ [0, 0],[0, 0]
↦
A = ivmatSet(A, 1, 2, bounds(1, 2));  A;
↦ [0, 0],[1, 2]
↦ [0, 0],[0, 0]
↦
```

D.8.8.6 unitMatrix

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `unitMatrix(n)`; `n int`

Return: `nxn` unit interval matrix

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
unitMatrix(2);
↦ [1, 1],[0, 0]
↦ [0, 0],[1, 1]
↦
unitMatrix(3);
↦ [1, 1],[0, 0],[0, 0]
↦ [0, 0],[1, 1],[0, 0]
↦ [0, 0],[0, 0],[1, 1]
↦
```

D.8.8.7 ivmatGaussian

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `ivmatGaussian(A)`; `A ivmat`

Return: 0, if `A` not invertible, `(1, Ainv)` if `A` invertible where `Ainv` is the inverse matrix

Purpose: Inverts an interval matrix using Gaussian elimination in the setting of interval arithmetic. Pivoting is handled as a special case as `I/I != [1,1]` and `I-I != [0,0]` in general.

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;
box B = list(bounds(7/8, 9/8), bounds(-1/10, 1/20));
ivmat J = evalJacobianAtBox (I, B); J;
↦ [69/20, 23/5],[-61/40, -27/40]
↦ [67/20, 24/5],[-159/40, -93/40]
↦
list result = ivmatGaussian(J);
ivmat Jinv = result[2];
Jinv;
↦ [1060/4273, 620/187],[-1220/561, -180/4273]
↦ [2680/12819, 1280/187],[-920/187, -3680/12819]
↦
Jinv * J;
↦ [-7657037/799051, 12073835/799051],[-3961862/799051, 6773506/799051]
↦ [-18293234/799051, 73172936/2397153],[-7807572/799051, 15513600/799051]
↦
```

D.8.8.8 evalPolyAtBox

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `evalPolyAtBox(f, B); f poly, B box`

Return: interval, evaluation of `f` at `B` using interval arithmetic

Purpose: computes an interval extension of the polynomial

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
poly f = x2+y-1;
box B = list(bounds(-1,1), bounds(1,3)/2);
interval I = evalPolyAtBox(f, B); I;
↦ [-1/2, 3/2]
```

D.8.8.9 evalJacobianAtBox

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `evalJacobianAtBox(I, B); I ideal, B box`

Return: Jacobian matrix of `I` where polynomials are evaluated at `B`

Purpose: evaluates each polynomial of the Jacobian matrix of `I` using interval arithmetic

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
ideal I = 2x2-xy+2y2-2, 2x2-3xy+3y2-2;
interval J = bounds(-1,1);
evalJacobianAtBox(I, list(J,J));
↦ [-5, 5],[-5, 5]
↦ [-7, 7],[-9, 9]
↦
```

D.8.8.10 rootIsolationNoPreprocessing

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `rootIsolationNoPreprocessing(I, B, eps);` `I ideal, B list of boxes, eps number;`

Assume: I is a zero-dimensional radical ideal

Return: `(L1, L2)`, where `L1` contains boxes smaller than eps which may contain an element of V(`I`), i.e. a root and `L2` contains boxes which contain exactly one element of V(`I`)

Purpose: Given input box(es) `start` we try to find all roots of `I` lying in `start` by computing boxes that contain exactly one root. If `eps` > 0 then boxes that become smaller than `eps` will be returned.

Theory: We first check for every box if it contains no roots by interval arithmetic. If this is inconclusive we apply the Newton step which, as outlined in [2] and [3], converges to a root lying in the starting box. If the result of the Newton step is already contained in the interior of the starting box, it contains a unique root.

Note: While `rootIsolation` does, `rootIsolationNoPreprocessing` does not check input ideal for necessary conditions.

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;  // V(I) has four elements
interval i = bounds(-3/2,3/2);
box B = list(i, i);
list result = rootIsolationNoPreprocessing(I, list(B), 1/512);
size(result[1]);
↦ 0
size(result[2]);
↦ 4
result;
↦ [1]:
↦    empty list
↦ [2]:
↦    [1]:
↦       [-9/16, -15/32] x [-129/128, -127/128]
↦    [2]:
↦       [15/32, 33/64] x [31/32, 129/128]
↦    [3]:
↦       [63/64, 33/32] x [-1/256, 1/32]
↦    [4]:
↦       [-129/128, -63/64] x [-1/256, 1/32]
```

D.8.8.11 rootIsolation

Procedure from library `rootisolation.lib` (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: `rootIsolation(I, [start, eps]); I ideal, start box, eps number`

Assume: I is a zero-dimensional radical ideal

Return: `(L1, L2)`, where `L1` contains boxes smaller than eps which may contain an element of V(`I`), i.e. a root and `L2` contains boxes which contain exactly one element of V(`I`)

Purpose: same as rootIsolationNoPreprocessing, but speeds up computation by preprocessing starting box

Theory: As every root of I is a root of the polynomials I[i], we use Groebner elimination to find univariate polynomials for every variable which have these roots as well. Using that I is zero-dimensional these Groebner bases may be quickly computed using FGLM. Applying root isolation to these univariate polynomials then provides smaller starting boxes which speed up computations in the multivariate case.

Note: This algorithm and some procedures used therein perform Groebner basis computations in basering. It is thus advised to define I w.r.t. a fast monomial ordering.
The algorithm performs checks on I to prevent errors. If I does not have the right number of generators, we first try to find a suitable Groebner basis. If this fails we apply the algorithm to the triangular decomposition of I.

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;  // V(I) has four elements
interval i = bounds(-3/2,3/2);
box B = list(i, i);
list result = rootIsolation(I, B);
result;
↦ [1]:
↦    empty list
↦ [2]:
↦    [1]:
↦       [-1, -1] x [0, 0]
↦    [2]:
↦       [-1/2, -1/2] x [-1, -1]
↦    [3]:
↦       [1/2, 1/2] x [1, 1]
↦    [4]:
↦       [1, 1] x [0, 0]
```

D.8.8.12 rootIsolationPrimdec

Procedure from library rootisolation.lib (see Section D.8.8 [rootisolation_lib], page 1889).

Usage: rootIsolationPrimdec(I); I ideal

Assume: I is a zero-dimensional radical ideal

Return: L, where L contains boxes which contain exactly one element of V(I)

Purpose: same as rootIsolation, but speeds up computation and improves output by doing a primary decomposition before doing the root isolation

Theory: For the primary decomposition we use the algorithm of Gianni-Traeger-Zarcharias.

Note: This algorithm and some procedures used therein perform Groebner basis computations in basering. It is thus advised to define I w.r.t. a fast monomial ordering.

Example:

```
LIB "rootisolation.lib";
ring R = 0,(x,y),dp;
```

```
ideal I = 2x2-xy+2y2-2,2x2-3xy+3y2-2;  // V(I) has four elements
list result = rootIsolationPrimdec(I);
result;
↦ [1]:
↦    [1, 1] x [0, 0]
↦ [2]:
↦    [1/2, 1/2] x [1, 1]
↦ [3]:
↦    [-1/2, -1/2] x [-1, -1]
↦ [4]:
↦    [-1, -1] x [0, 0]
```

D.8.9 signcond_lib

Library: signcond.lib

Purpose: Routines for computing realizable sign conditions

Author: Enrique A. Tobis, etobis@dc.uba.ar

Overview: Routines to determine the number of solutions of a multivariate polynomial system which satisfy a given sign configuration.

References:
Basu, Pollack, Roy, "Algorithms in Real Algebraic Geometry", Springer, 2003.

Procedures:

D.8.9.1 signcnd

Procedure from library `signcond.lib` (see Section D.8.9 [signcond_lib], page 1896).

Usage: signcnd(P,I); ideal P,I

Return: list: the sign conditions realized by the polynomials of P on V(I). The output of signcnd is a list of two lists. Both lists have the same length. This length is the number of sign conditions realized by the polynomials of P on the set V(i).
Each element of the first list indicates a sign condition of the polynomials of P.
Each element of the second list indicates how many elements of V(I) give rise to the sign condition expressed by the same position on the first list.
See the example for further explanations of the output.

Assume: I is a Groebner basis.

Note: The procedure psigncnd performs some pretty printing of this output.

Example:

```
LIB "signcond.lib";
ring r = 0,(x,y),dp;
ideal i = (x-2)*(x+3)*x,y*(y-1);
ideal P = x,y;
list l = signcnd(P,i);
size(l[1]);     // = the number of sign conditions of P on V(i)
↦ 6
//Each element of l[1] indicates a sign condition of the polynomials of P.
//The following means P[1] > 0, P[2] = 0:
l[1][2];
```

```
↦ [1]:
↦    1
↦ [2]:
↦    0
//Each element of l[2] indicates how many elements of V(I) give rise to
//the sign condition expressed by the same position on the first list.
//The following means that exactly 1 element of V(I) gives rise to the
//condition P[1] > 0, P[2] = 0:
l[2][2];
↦ 1
```

See also: Section D.8.9.3 [firstoct], page 1897; Section D.8.9.2 [psigncnd], page 1897.

D.8.9.2 psigncnd

Procedure from library `signcond.lib` (see Section D.8.9 [signcond_lib], page 1896).

Usage: psigncnd(P,l); ideal P, list l

Return: list: a formatted version of l

Example:

```
LIB "signcond.lib";
ring r = 0,(x,y),dp;
ideal i = (x-2)*(x+3)*x,(y-1)*(y+2)*(y+4);
ideal P = x,y;
list l = signcnd(P,i);
psigncnd(P,l);
↦ 1 elements of V(I) satisfy {P[1] = 0,P[2] > 0}
↦ 1 elements of V(I) satisfy {P[1] > 0,P[2] > 0}
↦ 1 elements of V(I) satisfy {P[1] < 0,P[2] > 0}
↦ 2 elements of V(I) satisfy {P[1] = 0,P[2] < 0}
↦ 2 elements of V(I) satisfy {P[1] > 0,P[2] < 0}
↦ 2 elements of V(I) satisfy {P[1] < 0,P[2] < 0}
↦ 
```

See also: Section D.8.9.1 [signcnd], page 1896.

D.8.9.3 firstoct

Procedure from library `signcond.lib` (see Section D.8.9 [signcond_lib], page 1896).

Usage: firstoct(I); I ideal

Return: number: the number of points of V(I) lying in the first octant

Assume: I is given by a Groebner basis.

Example:

```
LIB "signcond.lib";
ring r = 0,(x,y),dp;
ideal i = (x-2)*(x+3)*x,y*(y-1);
firstoct(i);
↦ 1
```

See also: Section D.8.9.1 [signcnd], page 1896.

D.8.10 zeroset_lib

Library: zeroset.lib

Purpose: Procedures for roots and factorization

Author: Thomas Bayer, email: tbayer@mathematik.uni-kl.de,
http://wwwmayr.informatik.tu-muenchen.de/personen/bayert/
Current address: Hochschule Ravensburg-Weingarten

Overview: Algorithms for finding the zero-set of a zero-dim. ideal in Q(a)[x_1,..,x_n], roots and factorization of univariate polynomials over Q(a)[t] where a is an algebraic number. Written in the scope of the diploma thesis (advisor: Prof. Gert-Martin Greuel) 'Computations of moduli spaces of semiquasihomogeneous singularities and an implementation in Singular'. This library is meant as a preliminary extension of the functionality of Singular for univariate factorization of polynomials over simple algebraic extensions in characteristic 0.

Note: Subprocedures with postfix 'Main' require that the ring contains a variable 'a' and no parameters, and the ideal 'mpoly', where 'minpoly' from the basering is stored.

Procedures:

D.8.10.1 Quotient

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: Quotient(f, g); where f,g are polynomials

Purpose: compute the quotient q and remainder r s.t. f = g*q + r, deg(r) < deg(g)

Return: list of polynomials
_[1] = quotient q
_[2] = remainder r

Assume: basering = Q[x] or Q(a)[x]

Note: This procedure is outdated, and should no longer be used. Use div and mod instead.

Example:

```
LIB "zeroset.lib";
ring R = (0,a), x, lp;
minpoly = a2+1;
poly f =  x4 - 2;
poly g = x - a;
list qr = Quotient(f, g);
qr;
↦ [1]:
↦    x3+(a)*x2-x+(-a)
↦ [2]:
↦    0
qr[1]*g + qr[2] - f;
↦ 1
```

D.8.10.2 remainder

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: remainder(f, g); where f,g are polynomials

Purpose: compute the remainder of the division of f by g, i.e. a polynomial r s.t. f = g*q + r, deg(r) < deg(g).

Return: poly

Assume: basering = Q[x] or Q(a)[x]

Note: outdated, use mod/reduce instead

Example:

```
LIB "zeroset.lib";
ring R = (0,a), x, lp;
minpoly = a2+1;
poly f =  x4 - 1;
poly g = x3 - 1;
remainder(f, g);
↦ x-1
```

D.8.10.3 roots

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: roots(f); where f is a polynomial

Purpose: compute all roots of f in a finite extension of QQ
without multiplicities.

Return: ring, a polynomial ring over an extension field of QQ, containing a list 'theRoots' and polynomials 'newA' and 'f':

- 'theRoots' is the list of roots of the polynomial f (no multiplicities)
- if the ground field is Q(a') and the extension field is Q(a), then
'newA' is the representation of a' in Q(a).
If the basering contains a parameter 'a' and the minpoly remains unchanged
then 'newA' = 'a'.
If the basering does not contain a parameter then 'newA' = 'a' (default).
- 'f' is the polynomial f in Q(a) (a' being substituted by 'newA')

Assume: ground field to be Q or a simple extension of Q given by a minpoly

Example:

```
LIB "zeroset.lib";
ring R = (0,a), x, lp;
minpoly = a2+1;
poly f = x3 - a;
def R1 = roots(f);
↦
↦ // 'roots' created a new ring which contains the list 'theRoots' and
↦ // the polynomials 'f' and 'newA'
↦ // To access the roots, newA and the new representation of f, type
↦    def R = roots(f); setring R; theRoots; newA; f;
↦
```

```
setring R1;
minpoly;
↦ (a4-a2+1)
newA;
↦ (a3)
f;
↦ x3+(-a3)
theRoots;
↦ [1]:
↦    (-a3)
↦ [2]:
↦    (a)
↦ [3]:
↦    (a3-a)
map F;
F[1] = theRoots[1];
F(f);
↦ 0
```

D.8.10.4 sqfrNorm

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: sqfrNorm(f); where f is a polynomial in Q(a)[x]

Purpose: compute the norm of the squarefree polynomial f in Q(a)[x].

Return: list with 3 entries

_[1] = squarefree norm of g (poly)
_[2] = g (= f(x - s*a)) (poly)
_[3] = s (int)

Assume: f must be squarefree, basering = Q(a)[x] and minpoly != 0.

Note: the norm is an element of Q[x]

Example:

```
LIB "zeroset.lib";
ring R = (0,a), x, lp;
minpoly = a2+1;
poly f =  x4 - 2*x + 1;
sqfrNorm(f);
↦ [1]:
↦    x8+4*x6-4*x5+8*x4+8*x3-4*x2+8*x+8
↦ [2]:
↦    x4+(-4a)*x3-6*x2+(4a-2)*x+(2a+2)
↦ [3]:
↦    1
```

D.8.10.5 zeroSet

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: zeroSet(I [,opt]); I=ideal, opt=integer

Purpose: compute the zero-set of the zero-dim. ideal I, in a finite extension of QQ.

Return: ring, a polynomial ring over an extension field of QQ, containing a list 'theZeroset', a polynomial 'newA', and an ideal 'id':

- 'theZeroset' is the list of the zeros of the ideal I, each zero is an ideal.
- if the ground field is Q(b) and the extension field is Q(a), then
  'newA' is the representation of b in Q(a).
  If the basering contains a parameter 'a' and the minpoly remains unchanged
  then 'newA' = 'a'.
  If the basering does not contain a parameter then 'newA' = 'a' (default).
- 'id' is the ideal I in Q(a)[x_1,...] (a' substituted by 'newA')

Assume: dim(I) = 0, and ground field to be Q or a simple extension of Q given by a minpoly.

Options: opt = 0: no primary decomposition (default)
opt > 0: primary decomposition

Note: If I contains an algebraic number (parameter) then I must be transformed w.r.t. 'newA' in the new ring.

Example:

```
LIB "zeroset.lib";
ring R = (0,a), (x,y,z), lp;
minpoly = a2 + 1;
ideal I = x2 - 1/2, a*z - 1, y - 2;
def T = zeroSet(I);
setring T;
minpoly;
↦ (4a4+4a2+9)
newA;
↦ (1/3a3+5/6a)
id;
↦ id[1]=(1/3a3+5/6a)*z-1
↦ id[2]=y-2
↦ id[3]=2*x2-1
theZeroset;
↦ [1]:
↦    _[1]=(-1/3a3+1/6a)
↦    _[2]=2
↦    _[3]=(-1/3a3-5/6a)
↦ [2]:
↦    _[1]=(1/3a3-1/6a)
↦    _[2]=2
↦    _[3]=(-1/3a3-5/6a)
map F1 = basering, theZeroset[1];
map F2 = basering, theZeroset[2];
F1(id);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
F2(id);
↦ _[1]=0
↦ _[2]=0
↦ _[3]=0
```

D.8.10.6 egcdMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: egcdMain(f, g); where f,g are polynomials in Q[a,x]

Purpose: compute the polynomial gcd of f and g over Q(a)[x]

Return: poly

Assume: basering = Q[x,a] and ideal mpoly is defined (it might be 0), this represents the ring Q(a)[x] together with its minimal polynomial.

Note: outdated, use gcd instead

D.8.10.7 factorMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: factorMain(f); where f is a polynomial

Purpose: compute the factorization of the squarefree polynomial f over Q(a)[t], minpoly = p(a).

Return: list with 2 entries
  _[1] = factors, first is a constant
  _[2] = multiplicities (not yet implemented)

Assume: basering = Q[x,a], representing Q(a)[x]. An ideal mpoly must be defined, representing the minimal polynomial (it might be 0!).

Note: outdated, use factorize instead

D.8.10.8 invertNumberMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: invertNumberMain(f); where f is a polynomial in Q[a]

Purpose: compute 1/f if f is a number in Q(a), i.e., f is represented by a polynomial in Q[a].

Return: poly 1/f

Assume: basering = Q[x_1,...,x_n,a], ideal mpoly must be defined and != 0 !

Note: outdated, use / instead

D.8.10.9 quotientMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: quotientMain(f, g); where f,g are polynomials in Q(a)[x]

Purpose: compute the quotient q and remainder r s.th. f = g*q + r, deg(r) < deg(g)

Return: list of polynomials
  _[1] = quotient q
  _[2] = remainder r

Assume: basering = Q[x,a] and ideal mpoly is defined (it might be 0), this represents the ring Q(a)[x] together with its minimal polynomial.

Note: outdated, use div/mod instead

D.8.10.10 remainderMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: remainderMain(f, g); where f,g are polynomials in Q[a,x]

Purpose: compute the remainder r s.t. f = g*q + r, deg(r) < deg(g)

Return: poly

Assume: basering = Q[x,a] and ideal mpoly is defined (it might be 0), this represents the ring Q(a)[x] together with its minimal polynomial.

Note: outdated, use mod/reduce instead

D.8.10.11 rootsMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: rootsMain(f); where f is a polynomial in Q[a,x]

Purpose: compute all roots of f in a finite extension of the QQ without multiplicities.

Return: list, all entries are polynomials

```
_[1] = roots of f, each entry is a polynomial
_[2] = 'newA' - if the ground field is Q(b) and the extension field
       is Q(a), then 'newA' is the representation of b in Q(a)
_[3] = minpoly of the algebraic extension of the ground field
```

Assume: basering = Q[x,a] ideal mpoly must be defined, it might be 0!

Note: might change the ideal mpoly!!

D.8.10.12 sqfrNormMain

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: sqfrNorm(f); where f is a polynomial in Q(a)[x]

Purpose: compute the norm of the squarefree polynomial f in Q(a)[x].

Return: list with 3 entries

```
_[1] = squarefree norm of g (poly)
_[2] = g (= f(x - s*a)) (poly)
_[3] = s (int)
```

Assume: f must be squarefree, basering = Q[x,a] and ideal mpoly is equal to 'minpoly', this represents the ring Q(a)[x] together with 'minpoly'.

Note: the norm is an element of Q[x]

D.8.10.13 containedQ

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: containedQ(data, f [, opt]); data=list; f=any type; opt=integer

Purpose: test if f is an element of data.

Return: int
0 if f not contained in data
1 if f contained in data

Options: opt = 0 : use '==' for comparing f with elements from data
opt = 1 : use `sameQ` for comparing f with elements from data

D.8.10.14 sameQ

Procedure from library `zeroset.lib` (see Section D.8.10 [zeroset_lib], page 1898).

Usage: sameQ(a, b); a,b=list/intvec

Purpose: test a == b elementwise, i.e., a[i] = b[i].

Return: int
0 if a != b
1 if a == b

D.9 Visualization

D.9.1 graphics_lib

Library: graphics.lib

Purpose: Procedures to use Graphics with Mathematica

Author: Christian Gorzel, gorzelc@math.uni-muenster.de

Procedures:

D.9.1.1 staircase

Procedure from library `graphics.lib` (see Section D.9.1 [graphics_lib], page 1904).

Usage: staircase(s,I); s a string, I ideal in two variables

Return: string with Mathematica input for displaying staircase diagrams of an ideal I, i.e. exponent vectors of the initial ideal of I

Note: ideal I should be given by a standard basis. Let s="" and copy and paste the result into a Mathematica notebook.

Example:

```
LIB "graphics.lib";
ring r0 = 0,(x,y),ls;
ideal I = -1x2y6-1x4y2, 7x6y5+1/2x7y4+6x4y6;
staircase("",std(I));
↦
↦ Show[Graphics[{
↦ {GrayLevel[0.5],Map[Rectangle[#,{9,9}] &, {{2,6},{6,2}}]},
↦ {PointSize[0.03], Map[Point,{{2,6},{6,2}}]},
↦ Table[Circle[{i,j},0.1],{i,0,9},{j,0,9}]},
↦   Axes->True,AspectRatio->Automatic]]
↦
ring r1 = 0,(x,y),dp;
ideal I = fetch(r0,I);
```

```
staircase("",std(I));
↦
↦ Show[Graphics[{
↦ {GrayLevel[0.5],Map[Rectangle[#,{12,9}] &, {{2,6},{7,4},{9,2}}]},
↦ {PointSize[0.03], Map[Point,{{2,6},{7,4},{9,2}}]},
↦ Table[Circle[{i,j},0.1],{i,0,12},{j,0,9}]},
↦   Axes->True,AspectRatio->Automatic]]
↦
ring r2 = 0,(x,y),wp(2,3);
ideal I = fetch(r0,I);
staircase("",std(I));
↦
↦ Show[Graphics[{
↦ {GrayLevel[0.5],Map[Rectangle[#,{13,9}] &, {{2,6},{8,3},{10,2},{6,5}}]},
↦ {PointSize[0.03], Map[Point,{{2,6},{8,3},{10,2},{6,5}}]},
↦ Table[Circle[{i,j},0.1],{i,0,13},{j,0,9}]},
↦   Axes->True,AspectRatio->Automatic]]
↦
// Paste the output into a Mathematica notebook
// active evaluation of the cell with SHIFT RETURN
```

D.9.1.2 mathinit

Procedure from library `graphics.lib` (see Section D.9.1 [graphics_lib], page 1904).

Usage: mathinit();

Return: initializing string for loading Mathematica's ImplicitPlot

Example:

```
LIB "graphics.lib";
mathinit();
↦ << Graphics‘ImplicitPlot‘
// Paste the output into a Mathematica notebook
// active evaluation of the cell with SHIFT RETURN
```

D.9.1.3 mplot

Procedure from library `graphics.lib` (see Section D.9.1 [graphics_lib], page 1904).

Usage: mplot(fname, I [,I1,I2,..,s]); fname=string; I,I1,I2,..=ideals, s=string representing the plot region.
Use the ideals I1,I2,.. in order to produce multiple plots (they need to have the same number of entries as I!).

Return: string, text with Mathematica commands to display a plot

Note: The plotregion is defaulted to -1,1 around zero.
For implicit given curves enter first the string returned by procedure mathinit into Mathematica in order to load ImplicitPlot. The following conventions for I are used:

- ideal with 2 entries in one variable means a parametrised plane curve,
- ideal with 3 entries in one variable means a parametrised space curve,
- ideal with 3 entries in two variables means a parametrised surface,
- ideal with 2 entries in two variables means an implicit curve
  given as I[1]==I[2],

- ideal with 1 entry (or one polynomial) in two variables means
  an implicit curve given as f == 0,

Example:

```
LIB "graphics.lib";
// ---------  plane curves ------------
ring rr0 = 0,x,dp; export rr0;
ideal I = x3 + x, x2;
ideal J = x2, -x+x3;
mplot("",I,J,"-2,2");
↦
↦ ParametricPlot[{{s^3+s,s^2},{s^2,s^3-s}},{s,-2,2},
↦  AspectRatio->Automatic];
↦
// Paste the output into a Mathematica notebook
// active evaluation of the cell with SHIFT RETURN
// --------- space curves --------------
I = x3,-1/10x3+x2,x2;
mplot("",I);
↦
↦ ParametricPlot3D[{{s^3,-1/10*s^3+s^2,s^2}},{s,-1,1},
↦  ViewPoint->{1.3,-2.4,2}];
↦
// Paste the output into a Mathematica notebook
// active evaluation of the cell with SHIFT RETURN
// ----------- surfaces -------------------
ring rr1 = 0,(x,y),dp; export rr1;
ideal J = xy,y,x2;
mplot("",J,"-2,1","1,2");
↦
↦ ParametricPlot3D[{{s*t,t,s^2}},{s,-2,1},{t,1,2},
↦  Boxed->True, Axes->True, ViewPoint->{1.3,-2.4,2}];
↦
// Paste the output into a Mathematica notebook
// active evaluation of the cell with SHIFT RETURN
kill rr0,rr1;
```

D.9.2 latex_lib

Library: latex.lib

Purpose: Typesetting of Singular-Objects in LaTeX2e

Author: Christian Gorzel, gorzelc@math.uni-muenster.de

Global variables:

TeXwidth, TeXnofrac, TeXbrack, TeXproj, TeXaligned, TeXreplace, NoDollars are used to control the typesetting. Call `texdemo();` to obtain a LaTeX2e file `texlibdemo.tex` explaining the features of `latex.lib` and its global variables.

`TeXwidth` (int) -1, 0, 1..9, >9: controls breaking of long polynomials
`TeXnofrac` (int) flag: write 1/2 instead of \frac{1}{2}
`TeXbrack` (string) "{", "(", "<", "|", empty string:
controls brackets around ideals and matrices

`TeXproj` (int) flag: write ":" instead of "," in vectors
`TeXaligned` (int) flag: write maps (and ideals) aligned
`TeXreplace` (list) list entries = 2 strings: replacing symbols
`NoDollars` (int) flag: suppresses surrounding $ signs

Procedures:

D.9.2.1 closetex

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: closetex(fname); fname string

Return: nothing; writes a LaTeX2e closing line into file `<fname>`.

Note: preceding ">>" are deleted and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
opentex("exmpl");
texobj("exmpl","{\\large \\bf hello}");
closetex("exmpl");
```

D.9.2.2 opentex

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: opentex(fname); fname string

Return: nothing; writes a LaTeX2e header into a new file `<fname>`.

Note: preceding ">>" are deleted and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
opentex("exmpl");
texobj("exmpl","hello");
closetex("exmpl");
```

D.9.2.3 tex

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: tex(fname); fname string

Return: nothing; calls latex (LaTeX2e) for compiling the file fname

Note: preceding ">>" are deleted and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
ring r;
ideal I = maxideal(7);
opentex("exp001");              // open latex2e document
texobj("exp001","An ideal ",I);
closetex("exp001");
tex("exp001");
↦ calling latex2e for: exp001.tex
↦
```

```
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2014/09/29 v1.4h Standard LaTeX document class
↦ (/usr/share/texmf-dist/tex/latex/base/size10.clo))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the `?' option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amssymb.sty
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦ No file exp001.aux.
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsb.fd) [1] (./exp001.aux) )
↦ Output written on exp001.dvi (1 page, 2912 bytes).
↦ Transcript written on exp001.log.
system("sh","rm exp001.*");
↦ 0
```

D.9.2.4 texdemo

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texdemo();

Return: nothing; generates a LaTeX2e file called `texlibdemo.tex` explaining the features of `latex.lib` and its global variables.

D.9.2.5 texfactorize

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texfactorize(fname,f); fname string, f poly

Return: if `fname=""`: string, f as a product of its irreducible factors
otherwise: append this string to the file `<fname>`, and return nothing.

Note: preceding ">>" are deleted and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
ring r2 = 13,(x,y),dp;
poly f = (x+1+y)^2*x3y*(2x-2y)*y12;
texfactorize("",f);
↦ $-2\cdot (-x+y)\cdot (x+y+1)^{2}\cdot x^{3}\cdot y^{13}$
ring R49 = (7,a),x,dp;
minpoly = a2+a+3;
poly f = (a24x5+x3)*a2x6*(x+1)^2;
f;
```

```
↦ (a+3)*x13+(2a-1)*x12+(-2a+1)*x10+(-a-3)*x9
texfactorize("",f);
↦ $(a+3)\cdot (x-1)\cdot (x+1)^{3}\cdot x^{9}$
```

D.9.2.6 texmap

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texmap(fname,m,@r1,@r2); fname string, m string/map, @r1,@r2 rings

Return: if `fname=""`: string, the map m from @r1 to @r2 (preceded by its name if m = string) in TeX-typesetting;
otherwise: append this string to the file `<fname>`, and return nothing.

Note: preceding `">>"` are deleted in `fname`, and suffix ".tex" (if not given) is added to `fname`.
If m is a string then it has to be the name of an existing map from @r1 to @r2.

Example:

```
LIB "latex.lib";
// -------- prepare for example ---------
if (defined(TeXaligned)) {int Teali=TeXaligned; kill TeXaligned;}
if (defined(TeXreplace)) {list Terep=TeXreplace; kill TeXreplace;}
// -------- the example starts here ---------
//
string fname = "tldemo";
ring @r1=0,(x,y,z),dp;
export @r1;
ring r2=0,(u,v),dp;
map @phi =(@r1,u2,uv -v,v2); export @phi;
list TeXreplace;
TeXreplace[1] = list("@phi","\\phi");    // @phi --> \phi
export TeXreplace;
texmap("","@phi",@r1,r2);                // standard form
↦ $$
↦ \begin{array}{rcc}
↦ \phi:\Q[x,y,z] & \longrightarrow & \Q[u,v]\\[2mm]
↦ \left(x,y,z\right) & \longmapsto &
↦  \left(
↦ \begin{array}{c}
↦ u^{2}\\
↦ uv-v\\
↦ v^{2}
↦ \end{array}
↦ \right)
↦ \end{array}
↦ $$
//
int TeXaligned; export TeXaligned;       // map in one line
texmap("",@phi,@r1,r2);
↦ $\Q[x,y,z]\longrightarrow\Q[u,v], \ \left(x,y,z\right)\longmapsto \left(u\
   ^{2},uv-v,v^{2}\right)$
//
kill @r1,TeXreplace,TeXaligned;
//
// --- restore global variables if previously defined ---
```

```
if (defined(Teali)) {int TeXaligned=Teali; export TeXaligned; kill Teali;}
if (defined(Terep)) {list TeXreplace=Terep; export TeXreplace; kill Terep;}
```

D.9.2.7 texname

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texname(fname,s); fname,s strings

Return: if `fname=""`: the transformed string s, for which the following rules apply:

```
      s’ + "~"              -->  "\\tilde{"+ s’ +"}"
     "_" + int              -->       "_{" + int +"}"
  "[" + s’ + "]"            -->      "_{" + s’ + "}"
   "A..Z" + int             --> "A..Z" + "^{" + int + "}"
   "a..z" + int             --> "a..z" + "_{" + int + "}"
"(" + int + "," + s’ + ")" --> "_{"+ int +"}" + "^{" + s’+"}"
```

Furthermore, strings which begin with a left brace are modified by deleting the first and the last character (which is then assumed to be a right brace).

if `fname!=""`: append the transformed string s to the file `<fname>`, and return nothing.

Note: preceding `">>"` are deleted in `fname`, and suffix `".tex"` (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
ring r = 0,(x,y),lp;
poly f = 3xy4 + 2xy2 + x5y3 + x + y6;
texname("","{f(10)}");
↦ f(10)
texname("","f(10) =");
↦ f_{10} =
texname("","n1");
↦ n_{1}
texname("","T1_12");
↦ T^{1}_{12}
texname("","g’_11");
↦ g’_{11}
texname("","f23");
↦ f_{23}
texname("","M[2,3]");
↦ M_{2,3}
texname("","A(0,3);");
↦ A_{0}^{3};
texname("","E~(3)");
↦ \tilde{E}_{3}
```

D.9.2.8 texobj

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texobj(fname,l); fname string, l list

Return: if `fname=""`: string, the entries of l in LaTeX-typesetting;
otherwise: append this string to the file `<fname>`, and return nothing.

Note: preceding `">>"` are deleted in `fname`, and suffix `".tex"` (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
// -------- prepare for example ---------
if (defined(TeXaligned)) {int Teali=TeXaligned; kill TeXaligned;}
if (defined(TeXbrack)){string Tebra=TeXbrack; kill TeXbrack;}
//
//  --------------  typesetting for polynomials ----------
ring r = 0,(x,y),lp;
poly f = x5y3 + 3xy4 + 2xy2 + y6;
f;
↦ x5y3+3xy4+2xy2+y6
texobj("",f);
↦ $$
↦ \begin{array}{rl}
↦ & x^{5}y^{3}+3xy^{4}+2xy^{2}+y^{6}\\
↦ \end{array}
↦ $$
//  --------------  typesetting for ideals ----------
ideal G = jacob(f);
G;
↦ G[1]=5x4y3+3y4+2y2
↦ G[2]=3x5y2+12xy3+4xy+6y5
texobj("",G);
↦ $$
↦ \left(
↦ \begin{array}{c}
↦ 5x^{4}y^{3}+3y^{4}+2y^{2}, \\
↦ 3x^{5}y^{2}+12xy^{3}+4xy+6y^{5}
↦ \end{array}
↦ \right)$$
//  --------------  variation of typesetting for ideals ----------
int TeXaligned = 1; export TeXaligned;
string TeXbrack = "<"; export TeXbrack;
texobj("",G);
↦ $\left<5x^{4}y^{3}+3y^{4}+2y^{2},3x^{5}y^{2}+12xy^{3}+4xy+6y^{5}\right>$
kill TeXaligned, TeXbrack;
//  --------------  typesetting for matrices ----------
matrix J = jacob(G);
texobj("",J);
↦ $$
↦ \left(
↦ \begin{array}{*{2}{c}}
↦ 20x^{3}y^{3} & 15x^{4}y^{2}+12y^{3}+4y \\
↦ 15x^{4}y^{2}+12y^{3}+4y & 6x^{5}y+36xy^{2}+4x+30y^{4}
↦ \end{array}
↦ \right)
↦ $$
//  --------------  typesetting for intmats ----------
intmat m[3][4] = 9,2,4,5,2,5,-2,4,-6,10,-1,2,7;
texobj("",m);
↦ $$
↦ \left(
↦ \begin{array}{*{4}{r}}
↦ 9 & 2 & 4 & 5\\
↦ 2 & 5 & -2 & 4\\
↦ -6 & 10 & -1 & 2
↦ \end{array}
↦ \right)
↦ $$
//
// --- restore global variables if previously defined ---
if (defined(Teali)){int TeXaligned=Teali; export TeXaligned; kill Teali;}
if (defined(Tebra)){string TeXbrack=Tebra; export TeXbrack; kill Tebra;}
```

D.9.2.9 texpoly

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texpoly(fname,p); fname string, p poly

Return: if `fname=""`: string, the polynomial p in LaTeX-typesetting;
otherwise: append this string to the file `<fname>`, and return nothing.

Note: preceding `">>"` are deleted in `fname`, and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
ring r0=0,(x,y,z),dp;
poly f = -1x^2 + 2;
texpoly("",f);
↦ $-x^{2}+2$
ring rr= real,(x,y,z),dp;
texpoly("",2x2y23z);
↦ $2.000x^{2}y^{23}z$
ring r7= 7,(x,y,z),dp;
poly f = 2x2y23z;
texpoly("",f);
↦ $2x^{2}y^{23}z$
ring rab =(0,a,b),(x,y,z),dp;
poly f = (-2a2 +b3 -2)/a * x2y4z5 + (a2+1)*x + a+1;
f;
↦ (-2a2+b3-2)/(a)*x2y4z5+(a2+1)*x+(a+1)
texpoly("",f);
↦ $-\frac{2a^{2}-b^{3}+2}{a}\cdot x^{2}y^{4}z^{5}+(a^{2}+1)\cdot x+(a+1)$
texpoly("",1/(a2+2)*x2+2/b*x+(a+1)/3);
↦ $\frac{1}{a^{2}+2}\cdot x^{2}+\frac{2}{b}\cdot x+\frac{a+1}{3}$
```

D.9.2.10 texproc

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texproc(fname,pname); fname,pname strings

Assume: ‘`pname`‘ is a procedure.

Return: if `fname=""`: string, the proc ‘`pname`‘ in a verbatim environment in LaTeX-typesetting;
otherwise: append this string to the file `<fname>`, and return nothing.

Note: preceding `">>"` are deleted in `fname`, and suffix ".tex" (if not given) is added to `fname`.

Example:

```
LIB "latex.lib";
texproc("","texproc");
↦ \begin{verbatim}
↦ proc texproc(string fname,string pname)
↦ {
↦   int i,j,k,nl;
↦   string @p,s,t;
↦ 
↦   j = 1;
↦ 
↦   if (defined(`pname`))
↦   { if (typeof(`pname`)=="proc")
↦     { @p = string(`pname`);
↦       nl = find(@p,newline);
↦       s = "\\begin{verbatim}" + newline;
↦       s = s + "proc " + pname + "(";
↦       i = find(@p,"parameter");       // collecting the parameters
↦       k = find(@p,"alias");           // and the alias arguments
↦       while((i and i < nl) or (k and k < nl))
↦       {
↦         if (i and (k==0  or i<k))
↦         {
↦          j=find(@p,";",i);
↦          t = @p[i+10,j-i-10];
↦          if(i>1){s = s + ",";};
↦          s = s + t;
↦         }
↦         if (k and (i==0  or k<i))
↦         {
↦          j=find(@p,";",k);
↦          t = @p[k,j-k];
↦          if(k>1){s = s + ",";};
↦          s = s + t;
↦         }
↦         i = find(@p,"parameter",j);
↦         k = find(@p,"alias",j);
↦       }
↦       s = s + ")" + newline;
↦      j++;                      // skip one for the newline
↦      i = find(@p,";"+"return();"+newline,j);
↦      if (!(i))
↦      { i = find(@p,";"+"RETURN();"+newline,j); }  // j kann hier weg
↦      s = s + "{" + @p[j,i-j-1] + "}" + newline;
↦      s = s + "\\end{verbatim}" + newline;
↦    }
↦   }
↦   else
↦   { print(" // -- Error: No such proc defined");
↦     return();
↦   }
↦   if(size(fname))
↦   { i=1;
↦      while (fname[i]==">"){i++;}
↦      fname = fname[i,size(fname)-i+1];
↦      if (size(fname)>=4)         // check if filename is ending with ".tex"
↦      { if(fname[size(fname)-3,4]!=".tex") {fname = fname +".tex"; }
↦      }
↦      else {fname = fname + ".tex";}
↦      write(fname,s);
↦    }
↦    else{return(s);}
↦ }
↦ \end{verbatim}
↦
```

D.9.2.11 texring

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: texring(fname, r[,L]); fname string, r ring, L list

Return: if `fname=""`: string, the ring in TeX-typesetting;
otherwise: append this string to the file `<fname>` and return nothing.

Note: preceding `">>"` are deleted and suffix `".tex"` (if not given) is added to `fname`.
The optional list L is assumed to be a list of strings which control, for instance the symbol for the field of coefficients.
For more details call `texdemo();` (generates a LaTeX2e file called `texlibdemo.tex` which explains all features of `texring`).

Example:

```
LIB "latex.lib";
ring r0 = 0,(x,y),dp;                // char = 0, polynomial ordering
texring("",r0);
↦ $\Q[x,y]$
//
ring r7 =7,(x(0..2)),ds;             // char = 7, local ordering
texring("",r7);
↦ $\Z_{7}[[x_{0},x_{1},x_{2}]]$
//
ring r1 = 0,(x1,x2,y1,y2),wp(1,2,3,4);
texring("",r1);
↦ $\Q[x_{1},x_{2},y_{1},y_{2}]$
//
ring rr = real,(x),dp;               // real numbers
texring("",rr);
↦ $\R[x]$
//
ring rC = complex,x,dp;              // complex coefficients
texring("",rC);
↦ $\C[x]$
//
ring rabc =(0,t1,t2,t3),(x,y),dp;    // ring with parameters
texring("",rabc);
↦ $\Q(t_{1},t_{2},t_{3})[x,y]$
//
ring ralg = (7,a),(x1,x2),ds;        // algebraic extension
```

```
minpoly = a2-a+3;
texring("",ralg);
↦ $\Z_{7}(a)[[x_{1},x_{2}]]$
texring("",ralg,"mipo");
↦ $\Z_{7}(a)/(a^{2}-a+3)[[x_{1},x_{2}]]$
//
ring r49=(49,a),x,dp;                // Galois field
texring("",r49);
↦ $\F_{49}[x]$
//
setring r0;                          // quotient ring
ideal i = x2-y3;
qring q = std(i);
texring("",q);
↦ $\Q[x,y]/\left(y^{3}-x^{2}\right)$
//
// ------------------ additional features --------------------
ring r9 =0,(x(0..9)),ds;
texring("",r9,1);
↦ $\Q[[x_{0},\ldots,x_{9}]]$
texring("",r9,"C","{","^G");
↦ $\C\{x_{0},x_{1},x_{2},x_{3},x_{4},x_{5},x_{6},x_{7},x_{8},x_{9}\}^G$
//
ring rxy = 0,(x(1..5),y(1..6)),ds;
intvec v = 5,6;
texring("",rxy,v);
↦ $\Q[[x_{1},\ldots,x_{5},y_{1},\ldots,y_{6}]]$
```

D.9.2.12 rmx

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: rmx(fname); fname string

Return: nothing; removes the `.log` and `.aux` files associated to the LaTeX file <fname>.

Note: If `fname` ends by `".dvi"` or `".tex"`, the `.dvi` or `.tex` file will be deleted, too.

Example:

```
LIB "latex.lib";
ring r;
poly f = x+y+z;
opentex("exp001");              // defaulted latex2e document
texobj("exp001","A polynom",f);
closetex("exp001");
tex("exp001");
↦ calling latex2e for: exp001.tex
↦
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2014/09/29 v1.4h Standard LaTeX document class
↦ (/usr/share/texmf-dist/tex/latex/base/size10.clo))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amssymb.sty
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦ No file exp001.aux.
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsb.fd) [1] (./exp001.aux) )
↦ Output written on exp001.dvi (1 page, 308 bytes).
↦ Transcript written on exp001.log.
rmx("exp001");   // removes aux and log file of exp001
system("sh","rm exp001.*");
↦ 0
```

D.9.2.13 xdvi

Procedure from library `latex.lib` (see Section D.9.2 [latex_lib], page 1906).

Usage: xdvi(fname[,style]); fname,style = string

Return: nothing; displays dvi-file fname.dvi with previewer xdvi

Note: suffix .dvi may be omitted in fname
style captures the program that will be called instead of the default (xdvi)

Example:

```
LIB "latex.lib";
intmat m[3][4] = 9,2,4,5,2,5,-2,4,-6,10,-1,2,7;
opentex("exp001");
texobj("exp001","An intmat:  ",m);
closetex("exp001");
tex("exp001");
↦ calling latex2e for: exp001.tex
↦
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2014/09/29 v1.4h Standard LaTeX document class
↦ (/usr/share/texmf-dist/tex/latex/base/size10.clo))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amssymb.sty
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦ No file exp001.aux.
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsb.fd) [1] (./exp001.aux) )
↦ Output written on exp001.dvi (1 page, 524 bytes).
↦ Transcript written on exp001.log.
xdvi("exp001");
↦ calling  xdvi  for : exp001
↦
system("sh","rm exp001.*");
↦ 0
```

D.9.3 surf_lib

Library: surf.lib

Purpose: Procedures for Graphics with Surf

Author: Hans Schoenemann, Frank Seelisch

Note: Using this library requires the program `surf` to be installed. You can download `surf` either from http://sourceforge.net/projects/surf or from ftp://jim.mathematik.uni-kl.de/pub/Math/Singular/utils/. The procedure surfer requires the program `surfer` (version 1.4.1 or newwer) to be installed. You can download `surfer` from http://imaginary.org/program/surfer

Procedures: See also: Section D.9.4 [surfex_lib], page 1918.

D.9.3.1 plot

Procedure from library `surf.lib` (see Section D.9.3 [surf_lib], page 1917).

Usage: plot(I); I ideal or poly

Assume: I defines a plane curve or a surface given by one equation

Return: nothing

Note: requires the external program ‘surf‘ to be installed,
to close the graphical interface just press ‘Q‘

Example:

```
LIB "surf.lib";
// ---------  plane curves ------------
ring rr0 = 0,(x1,x2),dp;
ideal I = x1^3 - x2^2;
plot(I);
ring rr1 = 0,(x,y,z),dp;
ideal I(1) = 2x2-1/2x3 +1-y+1;
plot(I(1));
//  ---- Singular Logo --------------
poly logo = ((x+3)^3 + 2*(x+3)^2 - y^2)*(x^3 - y^2)*((x-3)^3-2*(x-3)^2-y^2);
plot(logo);
// Steiner surface
```

```
ideal J(2) = x^2*y^2+x^2*z^2+y^2*z^2-17*x*y*z;
plot(J(2));
// ---------------------
plot(x*(x2-y2)+z2);
// E7
plot(x^3-x*y^3+z^2);
// Whitney umbrella
plot(z^2-x^2*y);
```

D.9.3.2 surfer

Procedure from library `surf.lib` (see Section D.9.3 [surf_lib], page 1917).

Usage: surfer(f); f poly

Assume: f defines a surface given by one equation

Return: nothing

Note: requires the external program ‘surfer‘ to be installed, to close the graphical interface just close the window of surfer

Example:

```
LIB "surf.lib";
ring rr1 = 0,(x,y,z),dp;
// Steiner surface
ideal J(2) = x^2*y^2+x^2*z^2+y^2*z^2-17*x*y*z;
surfer(J(2));
// ---------------------
surfer(x*(x2-y2)+z2);
// E7
surfer(x^3-x*y^3+z^2);
// Whitney umbrella
surfer(z^2-x^2*y);
```

D.9.4 surfex_lib

Library: surfex.lib

Purpose: Procedures for visualizing and rotating surfaces.

Author: Oliver Labs
This library uses the program surf
(written by Stefan Endrass and others)
and surfex (written by Oliver Labs and others, mainly Stephan Holzer).

Note: It is still an alpha version (see `http://www.AlgebraicSurface.net`)
This library requires the program surfex, surf and java to be installed. The software is used for producing raytraced images of surfaces. You can download `surfex` from `http://www.surfex.AlgebraicSurface.net`

surfex is a front-end for surf which aims to be easier to use than the original tool.

Procedures: See also: Section D.9.3 [surf_lib], page 1917.

D.9.4.1 plotRotated

Procedure from library `surfex.lib` (see Section D.9.4 [surfex_lib], page 1918).

Usage: plotRotated(poly p, list coords, list #)
This opens the external program surfex for drawing the surface given by p, seen as a surface in the real affine space with coordinates coords. The optional int parameter can be used to set plotting quality.

Assume: coords is a list of three variables.
The basering is of characteristic zero and without parameters.

Example:

```
LIB "surfex.lib";
"Example:";
// An easy example: a surface with four conical nodes.
ring r = 0, (x,y,z), dp;
poly cayley_cubic = x^3+y^3+z^3+1^3-1/4*(x+y+z+1)^3;
//     plotRotated(cayley_cubic, list(x,y,z));
// A difficult example: a surface with a one-dimensional real component!
poly whitney_umbrella = x^2*z-y^2;
// The Whitney Umbrella without its handle:
plotRotated(whitney_umbrella, list(x,y,z));
// The Whitney Umbrella together with its handle:
plotRotated(whitney_umbrella, list(x,y,z), 2);
```

D.9.4.2 plotRot

Procedure from library `surfex.lib` (see Section D.9.4 [surfex_lib], page 1918).

Usage: plotRot(poly p, list #)
Similar to plotRotated, but guesses automatically which coordinates should be used. The optional int parameter can be used to set plotting quality.

It opens the external program surfex for drawing the surface given by p, seen as a surface in the real affine space with coordinates coords.

Assume: The basering is of characteristic zero and without parameters.

Example:

```
LIB "surfex.lib";
"Example:";
// More variables in the basering, but only 3 variables in the polynomial:
ring r1 = 0, (w,x,y,z), dp;
poly cayley_cubic = x^3+y^3+z^3+1^3-1/4*(x+y+z+1)^3;
plotRot(cayley_cubic);
// Three variables in the basering, but fewer variables in the polynomial:
ring r2 = 0, (x,y,z), dp;
plotRot(x^2+y^2-1);
plotRot(y^2+z^2-1);
// A cubic surface with a solitary point:
// Use the additional parameter 3 to ask singular
// to compute the singular locus before calling surfex.
ring r3 = 0, (x,y,z), dp;
poly kn_10 = x^3-3*x*y^2+z^3+3*x^2+3*y^2+z^2;
plotRot(kn_10, 3);
```

```
// The swallowtail:
// a surface with a real solitary curve sticking out of the surface.
// Use the additional parameter 3 to ask singular
// to compute the singular locus before calling surfex.
poly swallowtail = -4*y^2*z^3-16*x*z^4+27*y^4+144*x*y^2*z+128*x^2*z^2-256*x^3;
```

D.9.4.3 plotRotatedList

Procedure from library `surfex.lib` (see Section D.9.4 [surfex_lib], page 1918).

Usage: plotRotatedList(list varieties, list coords, list #)
This opens the external program surfex for drawing the surfaces given by varieties, seen as a surface in the real affine space with coordinates coords. The optional int parameter can be used to set plotting quality.

Assume: coords is a list of three variables, varieties is a list of ideals describing the varieties to be shown.
The basering is of characteristic zero and without parameters.

Example:

```
LIB "surfex.lib";
"Example:";
// A cubic surface together with a tritangent plane
// (i.e. a plane which cuts out three lines).
ring r = 0, (x,y,z), dp;
poly cayley_cubic = x^3+y^3+z^3+1^3-1/4*(x+y+z+1)^3;
poly plane = 1-x-y-z;
plotRotatedList(list(cayley_cubic, plane), list(x,y,z));
// The same cubic and plane.
// The plane is not shown but only its intersection with the surface.
plotRotatedList(list(cayley_cubic, ideal(cayley_cubic, plane)), list(x,y,z));
```

D.9.4.4 plotRotatedDirect

Procedure from library `surfex.lib` (see Section D.9.4 [surfex_lib], page 1918).

Usage: plotRotatedDirect(list varieties, list #)
This opens the external program surfex for drawing the surfaces given by varieties, seen as a surface in the real affine space with coordinates x,y,z. The format for the list varieties is not fully documented yet; please, see the examples below and try to adjust the examples to suit your needs.
The optional int parameter can be used to set plotting quality.

Assume: Passes the equations directly to surfex, i.e., the variable names should be x,y,z.
The advantage is that one can use parameters p1, p2, ...; these will be passed to surfex.

Example:

```
LIB "surfex.lib";
"Example:";
// A cubic surface depending on a parameter:
ring r = (0,p1), (x,y,z), dp;
poly cayley_cubic = x^3+y^3+z^3+1^3-p1*(x+y+z+1)^3;
// The entries of the list of varieties can either be polynomials
plotRotatedDirect(list(list(list(cayley_cubic)),
list(),
list(list(1,"0.0","1.0","500","0.25+0.25*sin(PI*p1)"))
));
// or strings which represent surfex-readable polynomials
plotRotatedDirect(list(list(list("x^3+y^3+z^3+1^3-p1*(x+y+z+1)^3")),
list(),
list(list("1","0.0","1.0","500","0.25+0.25*sin(PI*p1)"))
));
// More complicated varieties
plotRotatedDirect(list(list(list("x^2+y^2-z^2-3^2"),
list("x*sin(p1)+y*cos(p1)-3")),
list(list(list(1,2))),
list(list("1","0.0","1.0","500","2*PI*p1"))
));
```

D.9.4.5 plotRotatedListFromSpecifyList

Procedure from library `surfex.lib` (see Section D.9.4 [surfex_lib], page 1918).

Usage: plotRotatedListFromSpecifyList(list varietiesList, list #); varietiesList has a complicated format (not documented yet); see the example.
The optional int parameter can be used to set plotting quality.

Assume: The basering is of characteristic zero.

Example:

```
LIB "surfex.lib";
"Example:";
// A cubic surface depending on a parameter:
ring r = (0,p1), (x,y,z), dp;
poly cayley_cubic = x^3+y^3+z^3+1^3-p1*(x+y+z+1)^3;
poly plane = 1-x-y-z;
plotRotatedListFromSpecifyList(list(list(list(list("eqno:","1"),
list("equation:",
string(cayley_cubic))
)
),
list(),
list(list(1,"0.0","1.0","500","0.25+0.25*sin(PI*p1)")),
list()
));
```

D.10 Coding theory

D.10.1 brnoeth_lib

Library: brnoeth.lib

Purpose: Brill-Noether Algorithm, Weierstrass-SG and AG-codes

Authors: Jose Ignacio Farran Martin, ignfar@eis.uva.es
Christoph Lossen, lossen@mathematik.uni-kl.de

Overview: Implementation of the Brill-Noether algorithm for solving the Riemann-Roch problem and applications to Algebraic Geometry codes. The computation of Weierstrass semigroups is also implemented.

The procedures are intended only for plane (singular) curves defined over a prime field of positive characteristic.
For more information about the library see the end of the file brnoeth.lib.

Procedures:

D.10.1.1 Adj_div

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: Adj_div(f [,l]); f a poly, [l a list]

Return: list L with the computed data:

L[1] a list of rings: L[1][1]=aff_r (affine), L[1][2]=Proj_R (projective),
L[2] an intvec with 2 entries (degree, genus),
L[3] a list of intvec (closed places),
L[4] an intvec (conductor),
L[5] a list of lists:
L[5][d][1] a (local) ring over an extension of degree d,
L[5][d][2] an intvec (degrees of base points of places of degree d)

Note: `Adj_div(f);` computes and stores the fundamental data of the plane curve defined by f as needed for AG codes.
In the affine ring you can find the following data:

poly CHI: affine equation of the curve,
ideal Aff_SLocus: affine singular locus (std),
list Inf_Points: points at infinity
Inf_Points[1]: singular points
Inf_Points[2]: non-singular points,
list Aff_SPoints: affine singular points (if not empty).

In the projective ring you can find the projective equation CHI of the curve (poly).
In the local rings L[5][d][1] you find:

list POINTS: base points of the places of degree d,
list LOC_EQS: local equations of the curve at the base points,
list BRANCHES: Hamburger-Noether developments of the places,
list PARAMETRIZATIONS: local parametrizations of the places,

Each entry of the list L[3] corresponds to one closed place (i.e., a place and all its conjugates) which is represented by an intvec of size two, the first entry is the degree of the place (in particular, it tells the local ring where to find the data describing one representative of the closed place), and the second one is the position of those data in the lists POINTS, etc., inside this local ring.
In the intvec L[4] (conductor) the i-th entry corresponds to the i-th entry in the list of places L[3].

With no optional arguments, the conductor is computed by local invariants of the singularities; otherwise it is computed by the Dedekind formula.
An affine point is represented by a list P where P[1] is std of a prime ideal and P[2] is an intvec containing the position of the places above P in the list of closed places L[3].
If the point is at infinity, P[1] is a homogeneous irreducible polynomial in two variables.

If `printlevel>=0` additional comments are displayed (default: `printlevel=0`).

Warning: The parameter of the needed field extensions is called 'a'. Thus, there should be no global object named 'a' when executing Adj_div.

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list C=Adj_div(y9+y8+xy6+x2y3+y2+x3);
↦ The genus of the curve is 3
def aff_R=C[1][1];      // the affine ring
setring aff_R;
listvar(aff_R);         // data in the affine ring
↦ // aff_R                           [0]  *ring
↦ // Inf_Points                      [0]  list, size: 2
↦ // Aff_SPoints                     [0]  list, size: 3
↦ // Aff_SLocus                      [0]  ideal (SB), 2 generator(s)
↦ // CHI                             [0]  poly
CHI;                    // affine equation of the curve
↦ x3+x2y3+xy6+y9+y8+y2
Aff_SLocus;             // ideal of the affine singular locus
↦ Aff_SLocus[1]=y8+y2
↦ Aff_SLocus[2]=x2+y6
Aff_SPoints[1];         // 1st affine singular point: (1:1:1), no.1
↦ [1]:
↦    _[1]=y
↦    _[2]=x
↦ [2]:
↦    1
Inf_Points[1];          // singular point(s) at infinity: (1:0:0), no.4
↦ [1]:
↦    [1]:
↦       y
↦    [2]:
↦       4
Inf_Points[2];          // list of non-singular points at infinity
↦ empty list
//
def proj_R=C[1][2];     // the projective ring
setring proj_R;
CHI;                    // projective equation of the curve
↦ x3z6+x2y3z4+xy6z2+y9+y8z+y2z7
C[2][1];                // degree of the curve
↦ 9
C[2][2];                // genus of the curve
↦ 3
C[3];                   // list of computed places
↦ [1]:
↦    1,1
↦ [2]:
↦    1,2
↦ [3]:
↦    2,1
↦ [4]:
↦    1,3
C[4];                   // adjunction divisor (all points are singular!)
↦ 2,2,2,42
//
// we look at the place(s) of degree 2 by changing to the ring
C[5][2][1];
↦ // coefficients: ZZ/2[a]/(...)
↦ // number of vars : 3
↦ //        block   1 : ordering ls
↦ //                  : names    x y t
↦ //        block   2 : ordering C
def S(2)=C[5][2][1];
setring S(2);
POINTS;                // base point(s) of place(s) of degree 2: (1:a:1)
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ (a)
↦    [3]:
↦ 1
LOC_EQS;               // local equation(s)
↦ [1]:
↦    y2+y3+(a+1)*y4+y6+(a+1)*y8+y9+(a)*xy2+(a+1)*xy4+xy6+(a+1)*x2y+(a)*x2y2\
  +x2y3+x3
PARAMETRIZATIONS;      // parametrization(s) and exactness
↦ [1]:
↦    [1]:
↦       _[1]=t2+(a+1)*t3
↦       _[2]=t3+(a+1)*t4
↦    [2]:
↦       3,4
BRANCHES;              // Hamburger-Noether development
↦ [1]:
↦    [1]:
↦       _[1,1]=0
↦       _[1,2]=x
↦       _[1,3]=0
↦       _[2,1]=0
↦       _[2,2]=1
↦       _[2,3]=(a+1)
↦    [2]:
↦       1,-4
↦    [3]:
↦       0
↦    [4]:
↦       y+(a+1)*xy+(a)*x2y+(a)*x2y2+(a+1)*x3+x3y+x3y3+(a)*x4+(a+1)*x4y2+(a+\
  1)*x4y3+x5+x5y2+(a)*x6+(a+1)*x6y2+x6y4+x6y5+x7y+(a+1)*x8+(a+1)*x8y+x8y4+(\
  a+1)*x8y6+x9+x9y7+(a+1)*x10+x11y6+(a+1)*x12y4+x13y5+x14+x14y+x15y4+x16+(a\
  +1)*x16y2+x17y3+x19y2+(a+1)*x20+x21y+x23
printlevel=plevel;
```

See also: Section D.10.1.2 [NSplaces], page 1925; Section D.10.1.10 [closed_points], page 1934.

D.10.1.2 NSplaces

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: NSplaces(h, CURVE), where h is an intvec and CURVE is a list

Return: list L with updated data of CURVE after computing all non-singular affine closed places whose degrees are in the intvec h:

in L[1][1]: (affine ring) lists Aff_Points(d) with affine non-singular
(closed) points of degree d (if non-empty),
in L[3]: the newly computed closed places are added,
in L[5]: local rings created/updated to store (repres. of) new places.

See Section D.10.1.1 [Adj_div], page 1922 for a description of the entries in L.

Note: The list_expression should be the output of the procedure Adj_div.
Raising `printlevel`, additional comments are displayed (default: `printlevel=0`).

Warning: The parameter of the needed field extensions is called 'a'. Thus, there should be no global object named 'a' when executing NSplaces.

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list C=Adj_div(x3y+y3+x);
↦ The genus of the curve is 3
// The list of computed places:
C[3];
↦ [1]:
↦    1,1
↦ [2]:
↦    1,2
// create places up to degree 4
list L=NSplaces(1..4,C);
// The list of computed places is now:
L[3];
↦ [1]:
↦    1,1
↦ [2]:
↦    1,2
↦ [3]:
↦    1,3
↦ [4]:
↦    2,1
↦ [5]:
↦    3,1
↦ [6]:
↦    3,2
↦ [7]:
↦    3,3
↦ [8]:
↦    3,4
↦ [9]:
↦    3,5
↦ [10]:
↦    3,6
↦ [11]:
↦    3,7
↦ [12]:
↦    4,1
↦ [13]:
↦    4,2
↦ [14]:
↦    4,3
// e.g., affine non-singular points of degree 4 :
def aff_r=L[1][1];
setring aff_r;
Aff_Points(4);
↦ [1]:
↦    [1]:
↦       _[1]=y2+y+1
↦       _[2]=x2+xy+x+1
↦    [2]:
↦       12
↦ [2]:
↦    [1]:
↦       _[1]=y4+y3+1
↦       _[2]=x+y3+y
↦    [2]:
↦       13
↦ [3]:
↦    [1]:
↦       _[1]=y4+y3+y2+y+1
↦       _[2]=x+y2+y+1
↦    [2]:
↦       14
// e.g., base point of the 1st place of degree 4 :
def S(4)=L[5][4][1];
setring S(4);
POINTS[1];
↦ [1]:
↦ (a3)
↦ [2]:
↦ (a2+a)
↦ [3]:
↦ 1
printlevel=plevel;
```

See also: Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.10 [closed_points], page 1934.

D.10.1.3 BrillNoether

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: BrillNoether(G,CURVE); G an intvec, CURVE a list

Return: list of ideals (each of them with two homogeneous generators, which represent the numerator, resp. denominator, of a rational function).
The corresponding rational functions form a vector space basis of the linear system L(G), G a rational divisor over a non-singular curve.

Note: The procedure must be called from the ring CURVE[1][2], where CURVE is the output of the procedure `NSplaces`.
The intvec G represents a rational divisor supported on the closed places of CURVE[3] (e.g. `G=2,0,-1;` means 2 times the closed place 1 minus 1 times the closed place 3).

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list C=Adj_div(x3y+y3+x);
↦ The genus of the curve is 3
C=NSplaces(1..4,C);
// the first 3 Places in C[3] are of degree 1.
// we define the rational divisor G = 4*C[3][1]+4*C[3][3] (of degree 8):
intvec G=4,0,4;
def R=C[1][2];
setring R;
list LG=BrillNoether(G,C);
↦ Vector basis successfully computed
// here is the vector basis of L(G):
LG;
↦ [1]:
↦    _[1]=1
↦    _[2]=1
↦ [2]:
↦    _[1]=y
↦    _[2]=x
↦ [3]:
↦    _[1]=z
↦    _[2]=x
↦ [4]:
↦    _[1]=y2
↦    _[2]=x2
↦ [5]:
↦    _[1]=xz2+y3
↦    _[2]=x3
↦ [6]:
↦    _[1]=xyz2+y4
↦    _[2]=x4
printlevel=plevel;
```

See also: Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.2 [NSplaces], page 1925; Section D.10.1.4 [Weierstrass], page 1927.

D.10.1.4 Weierstrass

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: Weierstrass(i, m, CURVE); i,m integers and CURVE a list

Return: list WS of two lists:
WS[1] list of integers (Weierstr. semigroup of the curve at place i up to m)
WS[2] list of ideals (the associated rational functions)

Note: The procedure must be called from the ring CURVE[1][2], where CURVE is the output of the procedure `NSplaces`.
i represents the place CURVE[3][i].
Rational functions are represented by numerator/denominator in form of ideals with two homogeneous generators.

Warning: The place must be rational, i.e., necessarily CURVE[3][i][1]=1.

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list C=Adj_div(x3y+y3+x);
↦ The genus of the curve is 3
C=NSplaces(1..4,C);
def R=C[1][2];
setring R;
// Place C[3][1] has degree 1 (i.e it is rational);
list WS=Weierstrass(1,7,C);
↦ Vector basis successfully computed
// the first part of the list is the Weierstrass semigroup up to 7 :
WS[1];
↦ [1]:
↦    0
↦ [2]:
↦    3
↦ [3]:
↦    5
↦ [4]:
↦    6
↦ [5]:
↦    7
// and the second part are the corresponding functions :
WS[2];
↦ [1]:
↦    _[1]=1
↦    _[2]=1
↦ [2]:
↦    _[1]=y
↦    _[2]=z
↦ [3]:
↦    _[1]=xy
↦    _[2]=z2
↦ [4]:
↦    _[1]=y2
↦    _[2]=z2
↦ [5]:
```

```
↦     _[1]=y3
↦     _[2]=xz2
printlevel=plevel;
```

See also: Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.3 [BrillNoether], page 1926; Section D.10.1.2 [NSplaces], page 1925.

D.10.1.5 extcurve

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: extcurve(d, CURVE); d an integer, CURVE a list

Return: list L which is the update of the list CURVE with additional entries

L[1][3]: ring (p,a),(x,y),lp (affine),
L[1][4]: ring (p,a),(x,y,z),lp (projective),
L[1][5]: ring (p,a),(x,y,t),ls (local),
L[2][3]: int (the number of rational places),

the rings being defined over a field extension of degree d.
If d<2 then `extcurve(d,CURVE);` creates a list L which is the update of the list CURVE with additional entries

L[1][5]: ring p,(x,y,t),ls,
L[2][3]: int (the number of computed places over the base field).

In both cases, in the ring L[1][5] lists with the data for all the computed rational places (after a field extension of degree d) are created (see Section D.10.1.1 [Adj_div], page 1922):

lists POINTS, LOC_EQS, BRANCHES, PARAMETRIZATIONS.

Note: The list CURVE should be the output of `NSplaces`, and must contain (at least) one place of degree d.
You actually need all the places with degree dividing d. Otherwise, not all the places are computed, but only part of them.
This procedure must be executed before constructing AG codes, even if no extension is needed. The ring L[1][4] must be active when constructing codes over the field extension.

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list C=Adj_div(x5+y2+y);
↦ The genus of the curve is 2
C=NSplaces(1..4,C);
// since we have all points up to degree 4, we can extend the curve
// to that extension, in order to get rational points over F_16;
C=extcurve(4,C);
↦ Total number of rational places : NrRatPl = 33
// e.g., display the basepoint of place no. 32:
def R=C[1][5];
```

```
setring R;
POINTS[32];
↦ [1]:
↦ (a3+a+1)
↦ [2]:
↦ (a+1)
↦ [3]:
↦ 1
printlevel=plevel;
```

See also: Section D.10.1.6 [AGcode_L], page 1930; Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.2 [NSplaces], page 1925; Section D.10.1.10 [closed_points], page 1934.

D.10.1.6 AGcode_L

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: AGcode_L(G, D, EC); G,D intvec, EC a list

Return: a generator matrix for the evaluation AG code defined by the divisors G and D.

Note: The procedure must be called within the ring EC[1][4], where EC is the output of `extcurve(d)` (or within the ring EC[1][2] if d=1).
The entry i in the intvec D refers to the i-th rational place in EC[1][5] (i.e., to POINTS[i], etc., see Section D.10.1.5 [extcurve], page 1929).
The intvec G represents a rational divisor (see Section D.10.1.3 [BrillNoether], page 1926 for more details).
The code evaluates the vector space basis of L(G) at the rational places given by D.

Warnings: G should satisfy $2 * genus - 2 < deg(G) < size(D)$, which is not checked by the algorithm.
G and D should have disjoint supports (checked by the algorithm).

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list HC=Adj_div(x3+y2+y);
↦ The genus of the curve is 1
HC=NSplaces(1..2,HC);
HC=extcurve(2,HC);
↦ Total number of rational places : NrRatPl = 9
def ER=HC[1][4];
setring ER;
intvec G=5;      // the rational divisor G = 5*HC[3][1]
intvec D=2..9;   // D = sum of the rational places no. 2..9 over F_4
// let us construct the corresponding evaluation AG code :
matrix C=AGcode_L(G,D,HC);
↦ Vector basis successfully computed
// here is a linear code of type [8,5,>=3] over F_4
print(C);
↦ 0,0,1,  1,    (a),  (a+1),(a+1),(a),
↦ 0,1,(a),(a+1),(a),  (a+1),(a),  (a+1),
↦ 1,1,1,  1,    1,    1,    1,    1,
↦ 0,0,1,  1,    (a+1),(a),  (a),  (a+1),
↦ 0,0,(a),(a+1),(a+1),(a),  1,    1
printlevel=plevel;
```

See also: Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.3 [BrillNoether], page 1926; Section D.10.1.5 [extcurve], page 1929.

D.10.1.7 AGcode_Omega

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: AGcode_Omega(G, D, EC); G,D intvec, EC a list

Return: a generator matrix for the residual AG code defined by the divisors G and D.

Note: The procedure must be called within the ring EC[1][4], where EC is the output of `extcurve(d)` (or within the ring EC[1][2] if d=1).
The entry i in the intvec D refers to the i-th rational place in EC[1][5] (i.e., to POINTS[i], etc., see Section D.10.1.5 [extcurve], page 1929).
The intvec G represents a rational divisor (see Section D.10.1.3 [BrillNoether], page 1926 for more details).
The code computes the residues of a vector space basis of
$\Omega(G-D)$ at the rational places given by D.

Warnings: G should satisfy $2*genus-2 < deg(G) < size(D)$, which is not checked by the algorithm.
G and D should have disjoint supports (checked by the algorithm).

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list HC=Adj_div(x3+y2+y);
↦ The genus of the curve is 1
HC=NSplaces(1..2,HC);
HC=extcurve(2,HC);
↦ Total number of rational places : NrRatPl = 9
def ER=HC[1][4];
setring ER;
intvec G=5;      // the rational divisor G = 5*HC[3][1]
intvec D=2..9;   // D = sum of the rational places no. 2..9 over F_4
// let us construct the corresponding residual AG code :
matrix C=AGcode_Omega(G,D,HC);
↦ Vector basis successfully computed
// here is a linear code of type [8,3,>=5] over F_4
print(C);
↦ 1,    0,    (a),(a+1),1,1,0,0,
↦ (a+1),(a),  1,  0,    1,0,1,0,
↦ (a+1),(a+1),(a),(a),  1,0,0,1
printlevel=plevel;
```

See also: Section D.10.1.6 [AGcode_L], page 1930; Section D.10.1.1 [Adj_div], page 1922; Section D.10.1.3 [BrillNoether], page 1926; Section D.10.1.5 [extcurve], page 1929.

D.10.1.8 prepSV

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: prepSV(G, D, F, EC); G,D,F intvecs and EC a list

Return: list E of size n+3, where n=size(D). All its entries but E[n+3] are matrices:

E[1]: parity check matrix for the current AG code

E[2] ... E[n+2]: matrices used in the procedure decodeSV

E[n+3]: intvec with

E[n+3][1]: correction capacity

epsilon

of the algorithm

E[n+3][2]: designed Goppa distance

delta

of the current AG code

Note: Computes the preprocessing for the basic (Skorobogatov-Vladut) decoding algorithm. The procedure must be called within the ring EC[1][4], where EC is the output of `extcurve(d)` (or in the ring EC[1][2] if d=1)
The intvec G and F represent rational divisors (see Section D.10.1.3 [BrillNoether], page 1926 for more details).
The intvec D refers to rational places (see Section D.10.1.7 [AGcode_Omega], page 1931 for more details.). The current AG code is `AGcode_Omega(G,D,EC)`.
If you know the exact minimum distance d and you want to use it in `decodeSV` instead of *delta* , you can change the value of E[n+3][2] to d before applying decodeSV.
If you have a systematic encoding for the current code and want to keep it during the decoding, you must previously permute D (using `permute_L(D,P);`), e.g., according to the permutation P=L[3], L being the output of `sys_code`.

Warnings: F must be a divisor with support disjoint from the support of D and with degree $epsilon + genus$, where

$epsilon := [(deg(G) - 3 * genus + 1)/2]$.
G should satisfy $2 * genus - 2 < deg(G) < size(D)$, which is not checked by the algorithm.
G and D should also have disjoint supports (checked by the algorithm).

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list HC=Adj_div(x3+y2+y);
↦ The genus of the curve is 1
HC=NSplaces(1..2,HC);
HC=extcurve(2,HC);
↦ Total number of rational places : NrRatPl = 9
def ER=HC[1][4];
setring ER;
```

```
intvec G=5;      // the rational divisor G = 5*HC[3][1]
intvec D=2..9;   // D = sum of the rational places no. 2..9 over F_4
// construct the corresp. residual AG code of type [8,3,>=5] over F_4:
matrix C=AGcode_Omega(G,D,HC);
↦ Vector basis successfully computed
// we can correct 1 error and the genus is 1, thus F must have degree 2
// and support disjoint from that of D;
intvec F=2;
list SV=prepSV(G,D,F,HC);
↦ Vector basis successfully computed
↦ Vector basis successfully computed
↦ Vector basis successfully computed
// now everything is prepared to decode with the basic algorithm;
// for example, here is a parity check matrix to compute the syndrome :
print(SV[1]);
↦ 0,0,1,  1,    (a),  (a+1),(a+1),(a),
↦ 0,1,(a),(a+1),(a),  (a+1),(a),  (a+1),
↦ 1,1,1,  1,    1,    1,    1,    1,
↦ 0,0,1,  1,    (a+1),(a),  (a),  (a+1),
↦ 0,0,(a),(a+1),(a+1),(a),  1,    1
// and here you have the correction capacity of the algorithm :
int epsilon=SV[size(D)+3][1];
epsilon;
↦ 1
printlevel=plevel;
```

See also: Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.9 [decodeSV], page 1933; Section D.10.1.5 [extcurve], page 1929; Section D.10.1.13 [permute_L], page 1936; Section D.10.1.12 [sys_code], page 1935.

D.10.1.9 decodeSV

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: decodeSV(y, K); y a row-matrix and K a list

Return: a codeword (row-matrix) if possible, resp. the 0-matrix (of size 1) if decoding is impossible.
For decoding the basic (Skorobogatov-Vladut) decoding algorithm is applied.

Note: The list_expression should be the output K of the procedure `prepSV`.
The matrix_expression should be a (1 x n)-matrix, where n = ncols(K[1]).
The decoding may fail if the number of errors is greater than the correction capacity of the algorithm.

Example:

```
LIB "brnoeth.lib";
int plevel=printlevel;
printlevel=-1;
ring s=2,(x,y),lp;
list HC=Adj_div(x3+y2+y);
↦ The genus of the curve is 1
HC=NSplaces(1..2,HC);
HC=extcurve(2,HC);
↦ Total number of rational places : NrRatPl = 9
```

```
def ER=HC[1][4];
setring ER;
intvec G=5;      // the rational divisor G = 5*HC[3][1]
intvec D=2..9;   // D = sum of the rational places no. 2..9 over F_4
// construct the corresp. residual AG code of type [8,3,>=5] over F_4:
matrix C=AGcode_Omega(G,D,HC);
↦ Vector basis successfully computed
// we can correct 1 error and the genus is 1, thus F must have degree 2
// and support disjoint from that of D
intvec F=2;
list SV=prepSV(G,D,F,HC);
↦ Vector basis successfully computed
↦ Vector basis successfully computed
↦ Vector basis successfully computed
// now we produce 1 error on the zero-codeword :
matrix y[1][8];
y[1,3]=a;
// and then we decode :
print(decodeSV(y,SV));
↦ 0,0,0,0,0,0,0,0
printlevel=plevel;
```

See also: Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.5 [extcurve], page 1929; Section D.10.1.8 [prepSV], page 1932.

D.10.1.10 closed_points

Procedure from library brnoeth.lib (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: closed_points(I); I an ideal

Return: list of prime ideals (each a Groebner basis), corresponding to the (distinct affine closed) points of V(I)

Note: The ideal must have dimension 0, the basering must have 2 variables, the ordering must be lp, and the base field must be finite and prime.
It might be convenient to set the option(redSB) in advance.

Example:

```
LIB "brnoeth.lib";
ring s=2,(x,y),lp;
// this is just the affine plane over F_4 :
ideal I=x4+x,y4+y;
list L=closed_points(I);
// and here you have all the points :
L;
↦ [1]:
↦    _[1]=y
↦    _[2]=x
↦ [2]:
↦    _[1]=y
↦    _[2]=x+1
↦ [3]:
↦    _[1]=y
↦    _[2]=x2+x+1
↦ [4]:
↦    _[1]=y+1
↦    _[2]=x
↦ [5]:
↦    _[1]=y+1
↦    _[2]=x+1
↦ [6]:
↦    _[1]=y+1
↦    _[2]=x2+x+1
↦ [7]:
↦    _[1]=y2+y+1
↦    _[2]=x
↦ [8]:
↦    _[1]=y2+y+1
↦    _[2]=x+1
↦ [9]:
↦    _[1]=y2+y+1
↦    _[2]=x+y
↦ [10]:
↦    _[1]=y2+y+1
↦    _[2]=x+y+1
```

See also: Section D.8.5 [triang_lib], page 1855.

D.10.1.11 dual_code

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: dual_code(G); G a matrix of numbers

Return: a generator matrix of the dual code generated by G

Note: The input should be a matrix G of numbers.
The output is also a parity check matrix for the code defined by G

Example:

```
LIB "brnoeth.lib";
ring s=2,T,lp;
// here is the Hamming code of length 7 and dimension 3
matrix G[3][7]=1,0,1,0,1,0,1,0,1,1,0,0,1,1,0,0,0,1,1,1,1;
print(G);
↦ 1,0,1,0,1,0,1,
↦ 0,1,1,0,0,1,1,
↦ 0,0,0,1,1,1,1
matrix H=dual_code(G);
print(H);
↦ 1,1,1,0,0,0,0,
↦ 1,0,0,1,1,0,0,
↦ 0,1,0,1,0,1,0,
↦ 1,1,0,1,0,0,1
```

D.10.1.12 sys_code

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: sys_code(C); C is a matrix of constants

Return: list L with:

L[1] is the generator matrix in standard form of an equivalent code,
L[2] is the parity check matrix in standard form of such code,
L[3] is an intvec which represents the needed permutation.

Note: Computes a systematic code which is equivalent to the given one.
The input should be a matrix of numbers.
The output has to be interpreted as follows: if the input was the generator matrix of an AG code then one should apply the permutation L[3] to the divisor D of rational points by means of `permute_L(D,L[3]);` before continuing to work with the code (for instance, if you want to use the systematic encoding together with a decoding algorithm).

Example:

```
LIB "brnoeth.lib";
ring s=3,T,lp;
matrix C[2][5]=0,1,0,1,1,0,1,0,0,1;
print(C);
↦ 0,1,0,1,1,
↦ 0,1,0,0,1
list L=sys_code(C);
L[3];
↦ 2,4,3,1,5
// here is the generator matrix in standard form
print(L[1]);
↦ 1,0,0,0,1,
↦ 0,1,0,0,0
// here is the control matrix in standard form
print(L[2]);
↦ 0, 0,1,0,0,
↦ 0, 0,0,1,0,
↦ -1,0,0,0,1
// we can check that both codes are dual to each other
print(L[1]*transpose(L[2]));
↦ 0,0,0,
↦ 0,0,0
```

See also: Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.13 [permute_L], page 1936; Section D.10.1.8 [prepSV], page 1932.

D.10.1.13 permute_L

Procedure from library `brnoeth.lib` (see Section D.10.1 [brnoeth_lib], page 1921).

Usage: permute_L(L, P); L,P either intvecs or lists

Return: list obtained from L by applying the permutation given by P.

Note: If P is a list, all entries must be integers.

Example:

```
LIB "brnoeth.lib";
list L=list();
L[1]="a";
L[2]="b";
```

```
L[3]="c";
L[4]="d";
intvec P=1,3,4,2;
// the list L is permuted according to P :
permute_L(L,P);
↦ [1]:
↦    a
↦ [2]:
↦    c
↦ [3]:
↦    d
↦ [4]:
↦    b
```

See also: Section D.10.1.7 [AGcode_Omega], page 1931; Section D.10.1.8 [prepSV], page 1932; Section D.10.1.12 [sys_code], page 1935.

D.10.2 decodegb_lib

Library: decodegb.lib

Purpose: Decoding and min distance of linear codes with GB

Author: Stanislav Bulygin, bulygin@mathematik.uni-kl.de

Overview: In this library we generate several systems used for decoding cyclic codes and finding their minimum distance. Namely, we work with the Cooper's philosophy and generalized Newton identities. The origindeal method of quadratic equations is worked out here as well. We also (for comparison) enable to work with the system of Fitzgerald-Lax. We provide some auxiliary functions for further manipulations and decoding. For an overview of the methods mentioned above Section C.8 [Decoding codes with Groebner bases], page 784. For the vanishing ideal computation the algorithm of Farr and Gao is implemented.

Procedures:

D.10.2.1 sysCRHT

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysCRHT(n,defset,e,q,m,[k]); n,e,q,m,k are int, defset list of int's

- n length of the cyclic code,
- defset is a list representing the defining set,
- e the error-correcting capacity,
- q field size
- m degree extension of the splitting field,
- if k>0 additional equations representing the fact that every two error positions are either different or at least one of them is zero

Return: the ring to work with the CRHT-ideal (with Sala's additions), containing an ideal with name 'crht'

Theory: Based on 'defset' of the given cyclic code, the procedure constructs the corresponding Cooper-Reed-Heleseth-Truong ideal 'crht'. With its help one can solve the decoding problem. For basics of the method Section C.8.2 [Cooper philosophy], page 786.

Example:

```
LIB "decodegb.lib";
// binary cyclic [15,7,5] code with defining set (1,3)
intvec v = option(get);
list defset=1,3;           // defining set
int n=15;                  // length
int e=2;                   // error-correcting capacity
int q=2;                   // basefield size
int m=4;                   // degree extension of the splitting field
int sala=1;                // indicator to add additional equations
def A=sysCRHT(n,defset,e,q,m);
setring A;
A;                         // shows the ring we are working in
↦ // coefficients: ZZ/2(a)
↦ // number of vars : 6
↦ //        block   1 : ordering lp
↦ //                  : names    Y(2) Y(1) Z(1) Z(2) X(2) X(1)
↦ //        block   2 : ordering C
print(crht);               // the CRHT-ideal
↦ Y(2)*Z(2)+Y(1)*Z(1)+X(1),
↦ Y(2)*Z(2)^3+Y(1)*Z(1)^3+X(2),
↦ X(1)^16+X(1),
↦ X(2)^16+X(2),
↦ Z(1)^16+Z(1),
↦ Z(2)^16+Z(2),
↦ Y(1)+1,
↦ Y(2)+1
option(redSB);
ideal red_crht=std(crht);  // reduced Groebner basis
print(red_crht);
↦ X(1)^16+X(1),
↦ X(2)*X(1)^15+X(2),
↦ X(2)^8+X(2)^4*X(1)^12+X(2)^2*X(1)^3+X(2)*X(1)^6,
↦ Z(2)^2*X(1)+Z(2)*X(1)^2+X(2)+X(1)^3,
↦ Z(2)^2*X(2)+Z(2)*X(2)*X(1)+X(2)^2*X(1)^14+X(2)*X(1)^2,
↦ Z(2)^16+Z(2),
↦ Z(1)+Z(2)+X(1),
↦ Y(1)+1,
↦ Y(2)+1
//===========================
A=sysCRHT(n,defset,e,q,m,sala);
setring A;
print(crht);                // CRHT-ideal with additional equations from Sala
↦ Y(2)*Z(2)+Y(1)*Z(1)+X(1),
↦ Y(2)*Z(2)^3+Y(1)*Z(1)^3+X(2),
↦ X(1)^16+X(1),
↦ X(2)^16+X(2),
↦ Z(1)^16+Z(1),
↦ Z(2)^16+Z(2),
↦ Y(1)+1,
↦ Y(2)+1,
↦ Z(1)^15*Z(2)+Z(1)^14*Z(2)^2+Z(1)^13*Z(2)^3+Z(1)^12*Z(2)^4+Z(1)^11*Z(2)^5+\
   Z(1)^10*Z(2)^6+Z(1)^9*Z(2)^7+Z(1)^8*Z(2)^8+Z(1)^7*Z(2)^9+Z(1)^6*Z(2)^10+Z\
   (1)^5*Z(2)^11+Z(1)^4*Z(2)^12+Z(1)^3*Z(2)^13+Z(1)^2*Z(2)^14+Z(1)*Z(2)^15
```

```
option(redSB);
ideal red_crht=std(crht);   // reduced Groebner basis
print(red_crht);
↦ X(1)^16+X(1),
↦ X(2)*X(1)^15+X(2),
↦ X(2)^8+X(2)^4*X(1)^12+X(2)^2*X(1)^3+X(2)*X(1)^6,
↦ Z(2)*X(1)^15+Z(2),
↦ Z(2)^2+Z(2)*X(1)+X(2)*X(1)^14+X(1)^2,
↦ Z(1)+Z(2)+X(1),
↦ Y(1)+1,
↦ Y(2)+1
red_crht[5];                // general error-locator polynomial for this code
↦ Z(2)^2+Z(2)*X(1)+X(2)*X(1)^14+X(1)^2
option(set,v);
```

See also: Section D.10.2.4 [sysBin], page 1944; Section D.10.2.3 [sysNewton], page 1941.

D.10.2.2 sysCRHTMindist

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysCRHTMindist(n,defset,w); n,w are int, defset is list of int's

- n length of the cyclic code,
- defset is a list representing the defining set,
- w is a candidate for the minimum distance

Return: the ring to work with the Sala's ideal for the minimum distance containing the ideal with name 'crht_md'

Theory: Based on 'defset' of the given cyclic code, the procedure constructs the corresponding Cooper-Reed-Heleseth-Truong ideal 'crht_md'. With its help one can find minimum distance of the code in the binary case. For basics of the method Section C.8.2 [Cooper philosophy], page 786.

Example:

```
LIB "decodegb.lib";
intvec v = option(get);
// binary cyclic [15,7,5] code with defining set (1,3)
list defset=1,3;             // defining set
int n=15;                    // length
int d=5;                     // candidate for the minimum distance
def A=sysCRHTMindist(n,defset,d);
setring A;
A;                           // shows the ring we are working in
↦ // coefficients: ZZ/2
↦ // number of vars : 5
↦ //        block   1 : ordering lp
↦ //                  : names    Z(1) Z(2) Z(3) Z(4) Z(5)
↦ //        block   2 : ordering C
print(crht_md);              // the Sala's ideal for mindist
↦ Z(1)+Z(2)+Z(3)+Z(4)+Z(5),
↦ Z(1)^3+Z(2)^3+Z(3)^3+Z(4)^3+Z(5)^3,
↦ Z(1)^15+1,
↦ Z(2)^15+1,
↦ Z(3)^15+1,
↦ Z(4)^15+1,
↦ Z(5)^15+1,
↦ Z(1)^15*Z(2)+Z(1)^14*Z(2)^2+Z(1)^13*Z(2)^3+Z(1)^12*Z(2)^4+Z(1)^11*Z(2)^5+\
   Z(1)^10*Z(2)^6+Z(1)^9*Z(2)^7+Z(1)^8*Z(2)^8+Z(1)^7*Z(2)^9+Z(1)^6*Z(2)^10+Z\
   (1)^5*Z(2)^11+Z(1)^4*Z(2)^12+Z(1)^3*Z(2)^13+Z(1)^2*Z(2)^14+Z(1)*Z(2)^15,
↦ Z(1)^15*Z(3)+Z(1)^14*Z(3)^2+Z(1)^13*Z(3)^3+Z(1)^12*Z(3)^4+Z(1)^11*Z(3)^5+\
   Z(1)^10*Z(3)^6+Z(1)^9*Z(3)^7+Z(1)^8*Z(3)^8+Z(1)^7*Z(3)^9+Z(1)^6*Z(3)^10+Z\
   (1)^5*Z(3)^11+Z(1)^4*Z(3)^12+Z(1)^3*Z(3)^13+Z(1)^2*Z(3)^14+Z(1)*Z(3)^15,
↦ Z(1)^15*Z(4)+Z(1)^14*Z(4)^2+Z(1)^13*Z(4)^3+Z(1)^12*Z(4)^4+Z(1)^11*Z(4)^5+\
   Z(1)^10*Z(4)^6+Z(1)^9*Z(4)^7+Z(1)^8*Z(4)^8+Z(1)^7*Z(4)^9+Z(1)^6*Z(4)^10+Z\
   (1)^5*Z(4)^11+Z(1)^4*Z(4)^12+Z(1)^3*Z(4)^13+Z(1)^2*Z(4)^14+Z(1)*Z(4)^15,
↦ Z(1)^15*Z(5)+Z(1)^14*Z(5)^2+Z(1)^13*Z(5)^3+Z(1)^12*Z(5)^4+Z(1)^11*Z(5)^5+\
   Z(1)^10*Z(5)^6+Z(1)^9*Z(5)^7+Z(1)^8*Z(5)^8+Z(1)^7*Z(5)^9+Z(1)^6*Z(5)^10+Z\
   (1)^5*Z(5)^11+Z(1)^4*Z(5)^12+Z(1)^3*Z(5)^13+Z(1)^2*Z(5)^14+Z(1)*Z(5)^15,
↦ Z(2)^15*Z(3)+Z(2)^14*Z(3)^2+Z(2)^13*Z(3)^3+Z(2)^12*Z(3)^4+Z(2)^11*Z(3)^5+\
   Z(2)^10*Z(3)^6+Z(2)^9*Z(3)^7+Z(2)^8*Z(3)^8+Z(2)^7*Z(3)^9+Z(2)^6*Z(3)^10+Z\
   (2)^5*Z(3)^11+Z(2)^4*Z(3)^12+Z(2)^3*Z(3)^13+Z(2)^2*Z(3)^14+Z(2)*Z(3)^15,
↦ Z(2)^15*Z(4)+Z(2)^14*Z(4)^2+Z(2)^13*Z(4)^3+Z(2)^12*Z(4)^4+Z(2)^11*Z(4)^5+\
   Z(2)^10*Z(4)^6+Z(2)^9*Z(4)^7+Z(2)^8*Z(4)^8+Z(2)^7*Z(4)^9+Z(2)^6*Z(4)^10+Z\
   (2)^5*Z(4)^11+Z(2)^4*Z(4)^12+Z(2)^3*Z(4)^13+Z(2)^2*Z(4)^14+Z(2)*Z(4)^15,
↦ Z(2)^15*Z(5)+Z(2)^14*Z(5)^2+Z(2)^13*Z(5)^3+Z(2)^12*Z(5)^4+Z(2)^11*Z(5)^5+\
   Z(2)^10*Z(5)^6+Z(2)^9*Z(5)^7+Z(2)^8*Z(5)^8+Z(2)^7*Z(5)^9+Z(2)^6*Z(5)^10+Z\
   (2)^5*Z(5)^11+Z(2)^4*Z(5)^12+Z(2)^3*Z(5)^13+Z(2)^2*Z(5)^14+Z(2)*Z(5)^15,
↦ Z(3)^15*Z(4)+Z(3)^14*Z(4)^2+Z(3)^13*Z(4)^3+Z(3)^12*Z(4)^4+Z(3)^11*Z(4)^5+\
   Z(3)^10*Z(4)^6+Z(3)^9*Z(4)^7+Z(3)^8*Z(4)^8+Z(3)^7*Z(4)^9+Z(3)^6*Z(4)^10+Z\
   (3)^5*Z(4)^11+Z(3)^4*Z(4)^12+Z(3)^3*Z(4)^13+Z(3)^2*Z(4)^14+Z(3)*Z(4)^15,
↦ Z(3)^15*Z(5)+Z(3)^14*Z(5)^2+Z(3)^13*Z(5)^3+Z(3)^12*Z(5)^4+Z(3)^11*Z(5)^5+\
   Z(3)^10*Z(5)^6+Z(3)^9*Z(5)^7+Z(3)^8*Z(5)^8+Z(3)^7*Z(5)^9+Z(3)^6*Z(5)^10+Z\
   (3)^5*Z(5)^11+Z(3)^4*Z(5)^12+Z(3)^3*Z(5)^13+Z(3)^2*Z(5)^14+Z(3)*Z(5)^15,
↦ Z(4)^15*Z(5)+Z(4)^14*Z(5)^2+Z(4)^13*Z(5)^3+Z(4)^12*Z(5)^4+Z(4)^11*Z(5)^5+\
   Z(4)^10*Z(5)^6+Z(4)^9*Z(5)^7+Z(4)^8*Z(5)^8+Z(4)^7*Z(5)^9+Z(4)^6*Z(5)^10+Z\
   (4)^5*Z(5)^11+Z(4)^4*Z(5)^12+Z(4)^3*Z(5)^13+Z(4)^2*Z(5)^14+Z(4)*Z(5)^15
option(redSB);
ideal red_crht_md=std(crht_md);
print(red_crht_md);           // reduced Groebner basis
↦ Z(5)^15+1,
↦ Z(4)^12+Z(4)^9*Z(5)^3+Z(4)^6*Z(5)^6+Z(4)^3*Z(5)^9+Z(5)^12,
↦ Z(3)^6+Z(3)^4*Z(4)*Z(5)+Z(3)^2*Z(4)^2*Z(5)^2+Z(3)*Z(4)^4*Z(5)+Z(3)*Z(4)*Z\
   (5)^4+Z(4)^6+Z(5)^6,
↦ Z(2)^2+Z(2)*Z(3)+Z(2)*Z(4)+Z(2)*Z(5)+Z(3)^5*Z(4)^10*Z(5)^2+Z(3)^5*Z(4)^9*\
   Z(5)^3+Z(3)^5*Z(4)^8*Z(5)^4+Z(3)^5*Z(4)^4*Z(5)^8+Z(3)^5*Z(4)^3*Z(5)^9+Z(3\
   )^5*Z(4)^2*Z(5)^10+Z(3)^4*Z(4)^11*Z(5)^2+Z(3)^4*Z(4)^8*Z(5)^5+Z(3)^4*Z(4)\
   ^5*Z(5)^8+Z(3)^4*Z(4)^2*Z(5)^11+Z(3)^3*Z(4)^10*Z(5)^4+Z(3)^3*Z(4)^9*Z(5)^\
   5+Z(3)^3*Z(4)^8*Z(5)^6+Z(3)^3*Z(4)^4*Z(5)^10+Z(3)^3*Z(4)^3*Z(5)^11+Z(3)^3\
   *Z(4)^2*Z(5)^12+Z(3)^3*Z(5)^14+Z(3)^2*Z(4)^11*Z(5)^4+Z(3)^2*Z(4)^8*Z(5)^7\
   +Z(3)^2*Z(4)^5*Z(5)^10+Z(3)^2*Z(4)^2*Z(5)^13+Z(3)^2*Z(4)*Z(5)^14+Z(3)^2+Z\
   (3)*Z(4)^10*Z(5)^6+Z(3)*Z(4)^9*Z(5)^7+Z(3)*Z(4)^8*Z(5)^8+Z(3)*Z(4)^4*Z(5)\
   ^12+Z(3)*Z(4)^3*Z(5)^13+Z(3)*Z(4)+Z(4)^11*Z(5)^6+Z(4)^8*Z(5)^9+Z(4)^5*Z(5\
   )^12+Z(4)^3*Z(5)^14+Z(4)^2,
↦ Z(1)+Z(2)+Z(3)+Z(4)+Z(5)
option(set,v);
```

D.10.2.3 sysNewton

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysNewton (n,defset,t,q,m,[tr]); n,t,q,m,tr int, defset is list int's
- n is length,
- defset is the defining set,
- t is the number of errors,
- q is basefield size,
- m is degree extension of the splitting field,
- if tr>0 it indicates that Newton identities in triangular form should be constructed

Return: the ring to work with the generalized Newton identities (in triangular form if applicable) containing the ideal with name 'newton'

Theory: Based on 'defset' of the given cyclic code, the procedure constructs the corresponding ideal 'newton' with the generalized Newton identities. With its help one can solve the decoding problem. For basics of the method Section C.8.3 [Generalized Newton identities], page 787.

Example:

```
LIB "decodegb.lib";
// Newton identities for a binary 3-error-correcting cyclic code of
//length 31 with defining set (1,5,7)
int n=31;          // length
list defset=1,5,7; //defining set
int t=3;           // number of errors
int q=2;           // basefield size
int m=5;           // degree extension of the splitting field
int tr=1;          // indicator of triangular form of Newton identities
def A=sysNewton(n,defset,t,q,m);
setring A;
A;                 // shows the ring we are working in
↦ // coefficients: ZZ/2(a)
↦ // number of vars : 34
↦ //        block   1 : ordering lp
↦ //                  : names    S(31) S(30) S(29) S(28) S(27) S(26) S(25) \
   S(24) S(23) S(22) S(21) S(20) S(19) S(18) S(17) S(16) S(15) S(14) S(13) S\
   (12) S(11) S(10) S(9) S(8) S(6) S(4) S(3) S(2) sigma(1) sigma(2) sigma(3)\
    S(7) S(5) S(1)
↦ //        block   2 : ordering C
print(newton);     // generalized Newton identities
↦ S(31)*sigma(1)+S(30)*sigma(2)+S(29)*sigma(3)+S(1),
↦ S(31)*sigma(2)+S(30)*sigma(3)+S(2)+sigma(1)*S(1),
↦ S(31)*sigma(3)+S(3)+S(2)*sigma(1)+sigma(2)*S(1),
↦ S(4)+S(3)*sigma(1)+S(2)*sigma(2)+sigma(3)*S(1),
↦ S(4)*sigma(1)+S(3)*sigma(2)+S(2)*sigma(3)+S(5),
↦ S(6)+S(4)*sigma(2)+S(3)*sigma(3)+sigma(1)*S(5),
↦ S(6)*sigma(1)+S(4)*sigma(3)+sigma(2)*S(5)+S(7),
↦ S(8)+S(6)*sigma(2)+sigma(1)*S(7)+sigma(3)*S(5),
↦ S(9)+S(8)*sigma(1)+S(6)*sigma(3)+sigma(2)*S(7),
↦ S(10)+S(9)*sigma(1)+S(8)*sigma(2)+sigma(3)*S(7),
↦ S(11)+S(10)*sigma(1)+S(9)*sigma(2)+S(8)*sigma(3),
↦ S(12)+S(11)*sigma(1)+S(10)*sigma(2)+S(9)*sigma(3),
↦ S(13)+S(12)*sigma(1)+S(11)*sigma(2)+S(10)*sigma(3),
↦ S(14)+S(13)*sigma(1)+S(12)*sigma(2)+S(11)*sigma(3),
↦ S(15)+S(14)*sigma(1)+S(13)*sigma(2)+S(12)*sigma(3),
↦ S(16)+S(15)*sigma(1)+S(14)*sigma(2)+S(13)*sigma(3),
↦ S(17)+S(16)*sigma(1)+S(15)*sigma(2)+S(14)*sigma(3),
↦ S(18)+S(17)*sigma(1)+S(16)*sigma(2)+S(15)*sigma(3),
↦ S(19)+S(18)*sigma(1)+S(17)*sigma(2)+S(16)*sigma(3),
↦ S(20)+S(19)*sigma(1)+S(18)*sigma(2)+S(17)*sigma(3),
↦ S(21)+S(20)*sigma(1)+S(19)*sigma(2)+S(18)*sigma(3),
↦ S(22)+S(21)*sigma(1)+S(20)*sigma(2)+S(19)*sigma(3),
↦ S(23)+S(22)*sigma(1)+S(21)*sigma(2)+S(20)*sigma(3),
↦ S(24)+S(23)*sigma(1)+S(22)*sigma(2)+S(21)*sigma(3),
↦ S(25)+S(24)*sigma(1)+S(23)*sigma(2)+S(22)*sigma(3),
↦ S(26)+S(25)*sigma(1)+S(24)*sigma(2)+S(23)*sigma(3),
↦ S(27)+S(26)*sigma(1)+S(25)*sigma(2)+S(24)*sigma(3),
↦ S(28)+S(27)*sigma(1)+S(26)*sigma(2)+S(25)*sigma(3),
↦ S(29)+S(28)*sigma(1)+S(27)*sigma(2)+S(26)*sigma(3),
↦ S(30)+S(29)*sigma(1)+S(28)*sigma(2)+S(27)*sigma(3),
↦ S(31)+S(30)*sigma(1)+S(29)*sigma(2)+S(28)*sigma(3),
↦ sigma(1)^32+sigma(1),
↦ sigma(2)^32+sigma(2),
↦ sigma(3)^32+sigma(3),
↦ S(2)+S(1)^2,
↦ S(4)+S(2)^2,
↦ S(6)+S(3)^2,
↦ S(8)+S(4)^2,
↦ S(10)+S(5)^2,
↦ S(12)+S(6)^2,
↦ S(14)+S(7)^2,
↦ S(16)+S(8)^2,
↦ S(18)+S(9)^2,
↦ S(20)+S(10)^2,
↦ S(22)+S(11)^2,
↦ S(24)+S(12)^2,
↦ S(26)+S(13)^2,
↦ S(28)+S(14)^2,
↦ S(30)+S(15)^2,
↦ S(16)^2+S(1),
↦ S(17)^2+S(3),
↦ S(18)^2+S(5),
↦ S(19)^2+S(7),
↦ S(20)^2+S(9),
↦ S(21)^2+S(11),
↦ S(22)^2+S(13),
↦ S(23)^2+S(15),
↦ S(24)^2+S(17),
↦ S(25)^2+S(19),
↦ S(26)^2+S(21),
↦ S(27)^2+S(23),
↦ S(28)^2+S(25),
↦ S(29)^2+S(27),
↦ S(30)^2+S(29),
↦ S(31)^2+S(31)
```

```
//================================
A=sysNewton(n,defset,t,q,m,tr);
setring A;
print(newton);     // generalized Newton identities in triangular form
↦ sigma(1)+S(1),
↦ S(2)+sigma(1)*S(1),
↦ S(3)+S(2)*sigma(1)+sigma(2)*S(1)+sigma(3),
↦ S(4)+S(3)*sigma(1)+S(2)*sigma(2)+sigma(3)*S(1),
↦ S(4)*sigma(1)+S(3)*sigma(2)+S(2)*sigma(3)+S(5),
↦ S(6)+S(4)*sigma(2)+S(3)*sigma(3)+sigma(1)*S(5),
↦ S(6)*sigma(1)+S(4)*sigma(3)+sigma(2)*S(5)+S(7),
↦ S(8)+S(6)*sigma(2)+sigma(1)*S(7)+sigma(3)*S(5),
↦ S(9)+S(8)*sigma(1)+S(6)*sigma(3)+sigma(2)*S(7),
↦ S(10)+S(9)*sigma(1)+S(8)*sigma(2)+sigma(3)*S(7),
↦ S(11)+S(10)*sigma(1)+S(9)*sigma(2)+S(8)*sigma(3),
↦ S(12)+S(11)*sigma(1)+S(10)*sigma(2)+S(9)*sigma(3),
↦ S(13)+S(12)*sigma(1)+S(11)*sigma(2)+S(10)*sigma(3),
↦ S(14)+S(13)*sigma(1)+S(12)*sigma(2)+S(11)*sigma(3),
↦ S(15)+S(14)*sigma(1)+S(13)*sigma(2)+S(12)*sigma(3),
↦ S(16)+S(15)*sigma(1)+S(14)*sigma(2)+S(13)*sigma(3),
↦ S(17)+S(16)*sigma(1)+S(15)*sigma(2)+S(14)*sigma(3),
↦ S(18)+S(17)*sigma(1)+S(16)*sigma(2)+S(15)*sigma(3),
↦ S(19)+S(18)*sigma(1)+S(17)*sigma(2)+S(16)*sigma(3),
↦ S(20)+S(19)*sigma(1)+S(18)*sigma(2)+S(17)*sigma(3),
↦ S(21)+S(20)*sigma(1)+S(19)*sigma(2)+S(18)*sigma(3),
↦ S(22)+S(21)*sigma(1)+S(20)*sigma(2)+S(19)*sigma(3),
↦ S(23)+S(22)*sigma(1)+S(21)*sigma(2)+S(20)*sigma(3),
↦ S(24)+S(23)*sigma(1)+S(22)*sigma(2)+S(21)*sigma(3),
↦ S(25)+S(24)*sigma(1)+S(23)*sigma(2)+S(22)*sigma(3),
↦ S(26)+S(25)*sigma(1)+S(24)*sigma(2)+S(23)*sigma(3),
↦ S(27)+S(26)*sigma(1)+S(25)*sigma(2)+S(24)*sigma(3),
↦ S(28)+S(27)*sigma(1)+S(26)*sigma(2)+S(25)*sigma(3),
↦ S(29)+S(28)*sigma(1)+S(27)*sigma(2)+S(26)*sigma(3),
↦ S(30)+S(29)*sigma(1)+S(28)*sigma(2)+S(27)*sigma(3),
↦ S(31)+S(30)*sigma(1)+S(29)*sigma(2)+S(28)*sigma(3),
↦ sigma(1)^32+sigma(1),
↦ sigma(2)^32+sigma(2),
↦ sigma(3)^32+sigma(3),
↦ S(2)+S(1)^2,
↦ S(4)+S(2)^2,
↦ S(6)+S(3)^2,
↦ S(8)+S(4)^2,
↦ S(10)+S(5)^2,
↦ S(12)+S(6)^2,
↦ S(14)+S(7)^2,
↦ S(16)+S(8)^2,
↦ S(18)+S(9)^2,
↦ S(20)+S(10)^2,
↦ S(22)+S(11)^2,
↦ S(24)+S(12)^2,
↦ S(26)+S(13)^2,
↦ S(28)+S(14)^2,
↦ S(30)+S(15)^2,
↦ S(16)^2+S(1),
↦ S(17)^2+S(3),
↦ S(18)^2+S(5),
↦ S(19)^2+S(7),
↦ S(20)^2+S(9),
↦ S(21)^2+S(11),
↦ S(22)^2+S(13),
↦ S(23)^2+S(15),
↦ S(24)^2+S(17),
↦ S(25)^2+S(19),
↦ S(26)^2+S(21),
↦ S(27)^2+S(23),
↦ S(28)^2+S(25),
↦ S(29)^2+S(27),
↦ S(30)^2+S(29),
↦ S(31)^2+S(31)
```

See also: Section D.10.2.4 [sysBin], page 1944; Section D.10.2.1 [sysCRHT], page 1937.

D.10.2.4 sysBin

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysBin (v,Q,n,[odd]); v,n,odd are int, Q is list of int's

- v a number if errors,
- Q is a defining set of the code,
- n the length,
- odd is an additional parameter: if
 set to 1, then the defining set is enlarged by odd elements,
 which are 2^(some power)*(some element in the def.set) mod n

Return: the ring with the resulting system called 'bin'

Theory: Based on Q of the given cyclic code, the procedure constructs the corresponding ideal 'bin' with the use of the Waring function. With its help one can solve the decoding problem. For basics of the method Section C.8.3 [Generalized Newton identities], page 787.

Example:

```
LIB "decodegb.lib";
// [31,16,7] quadratic residue code
list l=1,5,7,9,19,25;
// we do not need even synromes here
def A=sysBin(3,l,31);
setring A;
print(bin);
↦ S(1)+sigma(1),
↦ S(5)+sigma(1)^5+sigma(1)^3*sigma(2)+sigma(1)^2*sigma(3)+sigma(1)*sigma(2)\
   ^2+sigma(2)*sigma(3),
↦ S(7)+sigma(1)^7+sigma(1)^5*sigma(2)+sigma(1)^4*sigma(3)+sigma(1)^2*sigma(\
   2)*sigma(3)+sigma(1)*sigma(2)^3+sigma(1)*sigma(3)^2+sigma(2)^2*sigma(3),
↦ S(9)+sigma(1)^9+sigma(1)^7*sigma(2)+sigma(1)^6*sigma(3)+sigma(1)^5*sigma(\
   2)^2+sigma(1)^4*sigma(2)*sigma(3)+sigma(1)*sigma(2)^4+sigma(1)*sigma(2)*s\
   igma(3)^2+sigma(2)^3*sigma(3)+sigma(3)^3,
↦ S(19)+sigma(1)^19+sigma(1)^17*sigma(2)+sigma(1)^16*sigma(3)+sigma(1)^14*s\
   igma(2)*sigma(3)+sigma(1)^13*sigma(2)^3+sigma(1)^13*sigma(3)^2+sigma(1)^1\
   2*sigma(2)^2*sigma(3)+sigma(1)^11*sigma(2)^4+sigma(1)^9*sigma(2)^5+sigma(\
   1)^8*sigma(2)^4*sigma(3)+sigma(1)^8*sigma(2)*sigma(3)^3+sigma(1)^3*sigma(\
   2)^8+sigma(1)^2*sigma(2)*sigma(3)^5+sigma(1)*sigma(2)^9+sigma(1)*sigma(2)\
   ^3*sigma(3)^4+sigma(1)*sigma(3)^6+sigma(2)^8*sigma(3)+sigma(2)^5*sigma(3)\
   ^3+sigma(2)^2*sigma(3)^5,
↦ S(25)+sigma(1)^25+sigma(1)^23*sigma(2)+sigma(1)^22*sigma(3)+sigma(1)^21*s\
   igma(2)^2+sigma(1)^20*sigma(2)*sigma(3)+sigma(1)^17*sigma(2)^4+sigma(1)^1\
   7*sigma(2)*sigma(3)^2+sigma(1)^16*sigma(2)^3*sigma(3)+sigma(1)^16*sigma(3\
   )^3+sigma(1)^11*sigma(2)*sigma(3)^4+sigma(1)^10*sigma(3)^5+sigma(1)^9*sig\
   ma(2)^5*sigma(3)^2+sigma(1)^9*sigma(2)^2*sigma(3)^4+sigma(1)^8*sigma(2)^7\
   *sigma(3)+sigma(1)^8*sigma(2)^4*sigma(3)^3+sigma(1)^8*sigma(2)*sigma(3)^5\
   +sigma(1)^7*sigma(2)^9+sigma(1)^6*sigma(2)^8*sigma(3)+sigma(1)^5*sigma(2)\
   ^10+sigma(1)^4*sigma(2)^9*sigma(3)+sigma(1)*sigma(2)^12+sigma(1)*sigma(2)\
   ^9*sigma(3)^2+sigma(1)*sigma(3)^8+sigma(2)^11*sigma(3)+sigma(2)^8*sigma(3\
   )^3
```

See also: Section D.10.2.1 [sysCRHT], page 1937; Section D.10.2.3 [sysNewton], page 1941.

D.10.2.5 encode

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: encode (x, g); x a row vector (message), and g a generator matrix

Return: corresponding codeword

Example:

```
LIB "decodegb.lib";
ring r=2,x,dp;
matrix x[1][4]=1,0,1,0;
matrix g[4][7]=1,0,0,0,0,1,1,
0,1,0,0,1,0,1,
0,0,1,0,1,1,1,
0,0,0,1,1,1,0;
//encode x with the generator matrix g
print(encode(x,g));
↦ 1,0,1,0,1,0,0
```

D.10.2.6 syndrome

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: syndrome (h, c); h a check matrix, c a row vector (codeword)

Return: corresponding syndrome

Example:

```
LIB "decodegb.lib";
ring r=2,x,dp;
matrix x[1][4]=1,0,1,0;
matrix g[4][7]=1,0,0,0,0,1,1,
0,1,0,0,1,0,1,
0,0,1,0,1,1,1,
0,0,0,1,1,1,0;
//encode x with the generator matrix g
```

```
matrix c=encode(x,g);
// disturb
c[1,3]=0;
//compute syndrome
//corresponding check matrix
matrix check[3][7]=1,0,0,1,1,0,1,0,1,0,1,0,1,1,0,0,1,0,1,1,1;
print(syndrome(check,c));
↦ 0,
↦ 0,
↦ 1
c[1,3]=1;
//now c is a codeword
print(syndrome(check,c));
↦ 0,
↦ 0,
↦ 0
```

D.10.2.7 sysQE

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysQE(check,y,t,[fieldeq,formal]);check,y matrix;t,fieldeq,formal int

- check is a parity check matrix of the code
- y is a received word,
- t the number of errors to be corrected,
- if fieldeq=1, then field equations are added,
- if formal=0, field equations on (known) syndrome variables are not added, in order to add them (note that the exponent should be equal to the number of elements in the INITIAL alphabet) one needs to set formal>0 for the exponent

Return: the ring to work with together with the resulting system called 'qe'

Theory: Based on 'check' of the given linear code, the procedure constructs the corresponding ideal that gives an opportunity to compute unknown syndrome of the received word y. After computing the unknown syndromes one is able to solve the decoding problem. For basics of the method Section C.8.5 [Decoding method based on quadratic equations], page 789.

Example:

```
LIB "decodegb.lib";
intvec v = option(get);
//correct 2 errors in [7,3] 8-ary code RS code
int t=2; int q=8; int n=7; int redun=4;
ring r=(q,a),x,dp;
matrix h_full=genMDSMat(n,a);
matrix h=submat(h_full,1..redun,1..n);
matrix g=dual_code(h);
matrix x[1][3]=0,0,1,0;
matrix y[1][7]=encode(x,g);
//disturb with 2 errors
matrix rec[1][7]=errorInsert(y,list(2,4),list(1,a));
//generate the system
def A=sysQE(h,rec,t);
```

```
setring A;
print(qe);
↦ U(1)+a^3,
↦ U(2)+a^2,
↦ U(3)+a^6,
↦ U(4),
↦ V(1)*U(1)+V(2)*U(2)+U(3),
↦ V(1)*U(2)+V(2)*U(3)+U(4),
↦ V(1)*U(3)+V(2)*U(4)+U(5),
↦ V(1)*U(4)+V(2)*U(5)+U(6),
↦ V(1)*U(5)+V(2)*U(6)+U(7),
↦ V(1)*U(6)+V(2)*U(7)+U(1),
↦ V(2)*U(1)+V(1)*U(7)+U(2)
//let us decode
option(redSB);
ideal sys_qe=std(qe);
print(sys_qe);
↦ U(7)+a,
↦ U(6)+a^3,
↦ U(5)+a^3,
↦ U(4),
↦ U(3)+a^6,
↦ U(2)+a^2,
↦ U(1)+a^3,
↦ V(2)+1,
↦ V(1)+a^4
option(set,v);
```

See also: Section D.10.2.17 [sysFL], page 1959.

D.10.2.8 errorInsert

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: errorInsert(y,pos,val); y is matrix, pos,val are list of int's

- y is a (code) word,
- pos = positions where errors occurred,
- val = their corresponding values

Return: corresponding received word

Example:

```
LIB "decodegb.lib";
//correct 2 errors in [7,3] 8-ary code RS code
int t=2; int q=8; int n=7; int redun=4;
ring r=(q,a),x,dp;
matrix h_full=genMDSMat(n,a);
matrix h=submat(h_full,1..redun,1..n);
matrix g=dual_code(h);
matrix x[1][3]=0,0,1,0;
matrix y[1][7]=encode(x,g);
print(y);
↦ a6,a6,a3,a,0,0,1
//disturb with 2 errors
matrix rec[1][7]=errorInsert(y,list(2,4),list(1,a));
```

```
print(rec);
↦ a6,a2,a3,0,0,0,1
print(rec-y);
↦ 0,1,0,a,0,0,0
```

D.10.2.9 errorRand

Procedure from library decodegb.lib (see Section D.10.2 [decodegb_lib], page 1937).

Usage: errorRand(y, num, e); y is matrix, num,e are int
- y is a (code) word,
- num is the number of errors,
- e is an extension degree (if one wants values to be from GF(p^e))

Return: corresponding received word

Example:

```
LIB "decodegb.lib";
//correct 2 errors in [7,3] 8-ary code RS code
int t=2; int q=8; int n=7; int redun=4;
ring r=(q,a),x,dp;
matrix h_full=genMDSMat(n,a);
matrix h=submat(h_full,1..redun,1..n);
matrix g=dual_code(h);
matrix x[1][3]=0,0,1,0;
matrix y[1][7]=encode(x,g);
//disturb with 2 random errors
matrix rec[1][7]=errorRand(y,2,3);
print(rec);
↦ a3,a6,a2,a,0,0,1
print(rec-y);
↦ a4,0,a5,0,0,0,0
```

D.10.2.10 randomCheck

Procedure from library decodegb.lib (see Section D.10.2 [decodegb_lib], page 1937).

Usage: randomCheck(m, n, e); m,n,e are int
- m x n are dimensions of the matrix,
- e is an extension degree (if one wants values to be from GF(p^e))

Return: random check matrix

Example:

```
LIB "decodegb.lib";
int redun=5; int n=15;
ring r=2,x,dp;
//generate random check matrix for a [15,5] binary code
matrix h=randomCheck(redun,n,1);
print(h);
↦ 0,1,0,0,0,1,1,1,0,1,1,0,0,0,0,
↦ 1,1,0,0,0,0,0,1,0,0,0,1,0,0,0,
↦ 1,0,1,1,1,1,0,0,0,1,0,0,1,0,0,
↦ 1,1,0,1,1,0,0,0,0,1,0,0,0,1,0,
↦ 0,1,0,0,0,0,0,1,1,0,0,0,0,0,1
```

```
//corresponding generator matrix
matrix g=dual_code(h);
print(g);
↦ 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,0,1,1,0,0,0,0,0,0,0,0,
↦ 0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,
↦ 1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,
↦ 0,0,1,1,0,1,0,0,0,1,0,0,0,0,0,
↦ 0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,
↦ 1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,
↦ 0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,
↦ 0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,
↦ 1,1,0,0,0,1,0,0,0,0,0,0,0,0,1
```

D.10.2.11 genMDSMat

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: genMDSMat(n, a); n is int, a is number

- n x n are dimensions of the MDS matrix,
- a is a primitive element of the field.

Note: An MDS matrix is constructed in the following way. We take 'a' to be a generator of the multiplicative group of the field. Then we construct the Vandermonde matrix with this 'a'.

Assume: extension field should already be defined

Return: a matrix with the MDS property.

Example:

```
LIB "decodegb.lib";
int q=16; int n=15;
ring r=(q,a),x,dp;
//generate an MDS (Vandermonde) matrix
matrix h_full=genMDSMat(n,a);
print(h_full);
↦ 1,1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
↦ 1,a,  a2, a3, a4, a5, a6, a7, a8, a9, a10,a11,a12,a13,a14,
↦ 1,a2, a4, a6, a8, a10,a12,a14,a,  a3, a5, a7, a9, a11,a13,
↦ 1,a3, a6, a9, a12,1,  a3, a6, a9, a12,1,  a3, a6, a9, a12,
↦ 1,a4, a8, a12,a,  a5, a9, a13,a2, a6, a10,a14,a3, a7, a11,
↦ 1,a5, a10,1,  a5, a10,1,  a5, a10,1,  a5, a10,1,  a5, a10,
↦ 1,a6, a12,a3, a9, 1,  a6, a12,a3, a9, 1,  a6, a12,a3, a9,
↦ 1,a7, a14,a6, a13,a5, a12,a4, a11,a3, a10,a2, a9, a,  a8,
↦ 1,a8, a,  a9, a2, a10,a3, a11,a4, a12,a5, a13,a6, a14,a7,
↦ 1,a9, a3, a12,a6, 1,  a9, a3, a12,a6, 1,  a9, a3, a12,a6,
↦ 1,a10,a5, 1,  a10,a5, 1,  a10,a5, 1,  a10,a5, 1,  a10,a5,
↦ 1,a11,a7, a3, a14,a10,a6, a2, a13,a9, a5, a,  a12,a8, a4,
↦ 1,a12,a9, a6, a3, 1,  a12,a9, a6, a3, 1,  a12,a9, a6, a3,
↦ 1,a13,a11,a9, a7, a5, a3, a,  a14,a12,a10,a8, a6, a4, a2,
↦ 1,a14,a13,a12,a11,a10,a9, a8, a7, a6, a5, a4, a3, a2, a
```

See also: Section C.8.5 [Decoding method based on quadratic equations], page 789.

D.10.2.12 mindist

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: mindist (check, q); check matrix, q int

- check is a check matrix,
- q is the field size

Return: minimum distance of the code

Example:

```
LIB "decodegb.lib";
//determine a minimum distance for a [7,3] binary code
int q=8; int n=7; int redun=4; int t=redun+1;
ring r=(q,a),x,dp;
//generate random check matrix
matrix h=randomCheck(redun,n,1);
print(h);
↦ 0,1,0,1,0,0,0,
↦ 0,0,1,0,1,0,0,
↦ 1,1,0,0,0,1,0,
↦ 1,1,1,0,0,0,1
int l=mindist(h);
l;
↦ 3
```

D.10.2.13 decode

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: decode(check, rec, t); check, rec matrix, t int

- check is the check matrix of the code,
- rec is a received word,
- t is an upper bound for the number of errors one wants to correct

Note: The method described in Section C.8.5 [Decoding method based on quadratic equations], page 789 is used for decoding.

Assume: Errors in rec should be correctable, otherwise the output is unpredictable

Return: a codeword that is closest to rec

Example:

```
LIB "decodegb.lib";
//correct 1 error in [15,7] binary code
int t=1; int q=16; int n=15; int redun=10;
ring r=(q,a),x,dp;
//generate random check matrix
matrix h=randomCheck(redun,n,1);
matrix g=dual_code(h);
matrix x[1][n-redun]=0,0,1,0,1,0,1;
matrix y[1][n]=encode(x,g);
print(y);
↦ 1,0,1,0,1,0,1,1,1,1,0,0,1,0,1
// find out the minimum distance of the code
list l=mindist(h);
```

```
//disturb with errors
"Correct ",(l[1]-1) div 2," errors";
↦ Correct  1  errors
matrix rec[1][n]=errorRand(y,(l[1]-1) div 2,1);
print(rec);
↦ 1,0,1,0,1,0,1,1,1,0,0,0,1,0,1
//let us decode
matrix dec_word=decode(h,rec);
print(dec_word);
↦ 1,0,1,0,1,0,1,1,1,1,0,0,1,0,1
```

D.10.2.14 decodeRandom

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: decodeRandom(redun,q,ncodes,ntrials,[e]); all parameters int

- redun is a redundabcy of a (random) code,
- q is the field size,
- ncodes is the number of random codes to be processed,
- ntrials is the number of received vectors per code to be corrected
- If e is given it sets the correction capacity explicitly. It
should be used in case one expects some lower bound,
otherwise the procedure tries to compute the real minimum distance
to find out the error-correction capacity

Return: nothing;

Example:

```
LIB "decodegb.lib";
int q=32; int n=25; int redun=n-11; int t=redun+1;
ring r=(q,a),x,dp;
// correct 2 errors in 2 random binary codes, 3 trials each
decodeRandom(n,redun,2,3,2);
↦ check matrix:
↦ 0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
↦ 0,1,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
↦ 0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
↦ 1,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
↦ 0,1,1,1,0,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
↦ 0,0,1,0,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
↦ 0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
↦ 0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
↦ 1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
↦ The system is generated
↦ Codeword:
↦ 1,0,0,1,0,1,0,1,1,1,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0
↦ Received word:
↦ 1,0,0,1,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0
↦ The Groebenr basis of the QE system:
```

```
↦ U(25)+a^25,
↦ U(24)+a^20,
↦ U(23)+a^28,
↦ U(22)+a^7,
↦ U(21)+a^29,
↦ U(20)+a^19,
↦ U(19)+a^23,
↦ U(18)+a^19,
↦ U(17)+a^21,
↦ U(16)+a^9,
↦ U(15)+a^14,
↦ U(14)+a^25,
↦ U(13)+a^28,
↦ U(12)+a^14,
↦ U(11)+a^30,
↦ U(10)+a^27,
↦ U(9)+a^26,
↦ U(8)+a^7,
↦ U(7)+a^14,
↦ U(6)+a^15,
↦ U(5)+a^13,
↦ U(4)+a^7,
↦ U(3)+a^22,
↦ U(2)+a^11,
↦ U(1),
↦ V(2)+a^11,
↦ V(1)+a^24
↦ Codeword:
↦ 0,0,1,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,0,0,0,0,1,0,0
↦ Received word:
↦ 0,0,0,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,0,0,0,0,1,1,0
↦ The Groebenr basis of the QE system:
↦ U(25)+a^6,
↦ U(24)+a^16,
↦ U(23)+a^8,
↦ U(22)+a^2,
↦ U(21)+a^8,
↦ U(20)+a^13,
↦ U(19)+a,
↦ U(18)+a^12,
↦ U(17)+a^29,
↦ U(16)+a,
↦ U(15)+a^21,
↦ U(14)+a^16,
↦ U(13)+a^3,
↦ U(12)+a^4,
↦ U(11)+a^4,
↦ U(10)+a^16,
↦ U(9)+a^30,
↦ U(8)+a^26,
↦ U(7)+a^17,
↦ U(6)+a^2,
↦ U(5)+a^15,
↦ U(4)+a^24,
↦ U(3)+a^23,
↦ U(2)+a^27,
↦ U(1),
↦ V(2)+a^27,
↦ V(1)+a^25
↦ Codeword:
↦ 0,0,0,1,0,1,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ Received word:
↦ 0,0,0,1,1,1,0,1,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ The Groebenr basis of the QE system:
↦ U(25)+a^7,
↦ U(24)+a^21,
↦ U(23)+a^5,
↦ U(22)+a^9,
↦ U(21)+a^21,
↦ U(20)+a^11,
↦ U(19)+a^22,
↦ U(18)+a^14,
↦ U(17)+a,
↦ U(16)+a^11,
↦ U(15)+a^13,
↦ U(14)+a^10,
↦ U(13)+a^19,
↦ U(12)+a^18,
↦ U(11)+a^26,
↦ U(10)+a^11,
↦ U(9)+a^16,
↦ U(8)+a^22,
↦ U(7)+a^25,
↦ U(6)+a^13,
↦ U(5)+a^8,
↦ U(4)+a^28,
↦ U(3)+a^4,
↦ U(2)+a^2,
↦ U(1),
↦ V(2)+a^2,
↦ V(1)+a^11
↦ check matrix:
↦ 0,1,0,1,1,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,0,1,1,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,1,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
↦ 1,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
↦ 0,1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
↦ 1,0,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
↦ 0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
↦ 1,0,1,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
↦ 1,0,0,1,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
↦ 1,0,0,1,0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
↦ 1,0,0,0,1,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1
↦ The system is generated
↦ Codeword:
↦ 0,0,1,0,1,0,1,0,1,0,0,1,1,0,0,0,0,0,1,1,0,0,0,1,0
↦ Received word:
↦ 0,0,1,0,1,1,1,0,1,0,0,0,1,0,0,0,0,0,1,1,0,0,0,1,0
↦ The Groebenr basis of the QE system:
↦ U(25)+a^4,
↦ U(24)+a^4,
↦ U(23)+a^6,
↦ U(22)+a^17,
↦ U(21)+a^13,
↦ U(20)+a^27,
↦ U(19)+a^21,
↦ U(18)+a^8,
↦ U(17)+a^16,
↦ U(16)+a^8,
↦ U(15)+a^15,
↦ U(14)+a^12,
↦ U(13)+a^2,
↦ U(12)+a^3,
↦ U(11)+a^22,
↦ U(10)+a^26,
↦ U(9)+a^8,
↦ U(8)+a^23,
↦ U(7)+a,
↦ U(6)+a^11,
↦ U(5)+a^4,
↦ U(4)+a^16,
↦ U(3)+a^2,
↦ U(2)+a,
↦ U(1),
↦ V(2)+a,
↦ V(1)+a^16
↦ Codeword:
↦ 1,0,0,1,1,1,1,1,0,0,0,0,0,1,1,0,1,0,1,0,1,1,0,0,0
↦ Received word:
↦ 0,0,0,1,1,0,1,1,0,0,0,0,0,1,1,0,1,0,1,0,1,1,0,0,0
↦ The Groebenr basis of the QE system:
↦ U(25)+a^6,
↦ U(24)+a^7,
↦ U(23)+a^30,
↦ U(22)+a^23,
↦ U(21)+a^22,
↦ U(20)+a^5,
↦ U(19)+a^26,
↦ U(18)+a^12,
↦ U(17)+a,
↦ U(16)+a^14,
↦ U(15)+a^20,
↦ U(14)+a^29,
↦ U(13)+a^3,
↦ U(12)+a^15,
↦ U(11)+a^11,
↦ U(10)+a^13,
↦ U(9)+a^16,
↦ U(8)+a^10,
↦ U(7)+a^17,
↦ U(6)+a^21,
↦ U(5)+a^8,
↦ U(4)+a^24,
↦ U(3)+a^4,
↦ U(2)+a^2,
↦ U(1),
↦ V(2)+a^2,
↦ V(1)+a^5
↦ Codeword:
↦ 1,0,1,1,1,1,0,0,0,1,1,1,1,0,1,0,0,1,0,0,0,0,0,1,1
↦ Received word:
↦ 1,0,1,1,0,1,0,0,0,1,1,1,0,0,1,0,0,1,0,0,0,0,0,1,1
↦ The Groebenr basis of the QE system:
↦ U(25)+a^30,
↦ U(24)+a^2,
↦ U(23)+a^20,
↦ U(22)+a^5,
↦ U(21)+a^20,
↦ U(20)+a^9,
↦ U(19)+a^18,
↦ U(18)+a^29,
↦ U(17)+a^12,
↦ U(16)+a^4,
↦ U(15)+a^5,
↦ U(14)+a^9,
↦ U(13)+a^15,
↦ U(12)+a^10,
↦ U(11)+a^10,
↦ U(10)+a^9,
↦ U(9)+a^6,
↦ U(8)+a^18,
↦ U(7)+a^23,
↦ U(6)+a^5,
↦ U(5)+a^3,
↦ U(4)+a^27,
↦ U(3)+a^17,
↦ U(2)+a^24,
↦ U(1),
↦ V(2)+a^24,
↦ V(1)+a^16
```

D.10.2.15 decodeCode

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: decodeCode(check, ntrials, [e]); check matrix, ntrials,e int

- check is a parity check matrix for the code,
- ntrials is the number of received vectors per code to be corrected.

- If e is given it sets the correction capacity explicitly. It
should be used in case one expects some lower bound,
otherwise the procedure tries to compute the real minimum distance
to find out the error-correction capacity

Return: nothing;

Example:

```
LIB "decodegb.lib";
int q=32; int n=25; int redun=n-11; int t=redun+1;
ring r=(q,a),x,dp;
matrix check=randomCheck(redun,n,1);
// correct 2 errors in using the code above, 3 trials
decodeCode(check,3,2);
↦ check matrix:
↦ 0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
↦ 0,1,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
↦ 0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
↦ 1,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
↦ 0,1,1,1,0,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
↦ 0,0,1,0,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
↦ 0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
↦ 0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
↦ 1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
↦ The system is generated
↦ Codeword:
↦ 1,0,0,1,0,1,0,1,1,1,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0
↦ Received word:
↦ 1,0,0,1,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0
↦ Groebner basis of the QE system:
↦ U(25)+a^25,
↦ U(24)+a^20,
↦ U(23)+a^28,
↦ U(22)+a^7,
↦ U(21)+a^29,
↦ U(20)+a^19,
↦ U(19)+a^23,
↦ U(18)+a^19,
↦ U(17)+a^21,
↦ U(16)+a^9,
↦ U(15)+a^14,
↦ U(14)+a^25,
↦ U(13)+a^28,
↦ U(12)+a^14,
↦ U(11)+a^30,
↦ U(10)+a^27,
↦ U(9)+a^26,
↦ U(8)+a^7,
↦ U(7)+a^14,
↦ U(6)+a^15,
↦ U(5)+a^13,
↦ U(4)+a^7,
↦ U(3)+a^22,
↦ U(2)+a^11,
↦ U(1),
↦ V(2)+a^11,
↦ V(1)+a^24
↦ Codeword:
↦ 0,0,1,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,0,0,0,0,1,0,0
↦ Received word:
↦ 0,0,0,1,1,1,1,0,0,1,0,1,0,1,1,0,1,1,0,0,0,0,1,1,0
↦ Groebner basis of the QE system:
↦ U(25)+a^6,
↦ U(24)+a^16,
↦ U(23)+a^8,
↦ U(22)+a^2,
↦ U(21)+a^8,
↦ U(20)+a^13,
↦ U(19)+a,
↦ U(18)+a^12,
↦ U(17)+a^29,
↦ U(16)+a,
↦ U(15)+a^21,
↦ U(14)+a^16,
↦ U(13)+a^3,
↦ U(12)+a^4,
↦ U(11)+a^4,
↦ U(10)+a^16,
↦ U(9)+a^30,
↦ U(8)+a^26,
↦ U(7)+a^17,
↦ U(6)+a^2,
↦ U(5)+a^15,
↦ U(4)+a^24,
↦ U(3)+a^23,
↦ U(2)+a^27,
↦ U(1),
↦ V(2)+a^27,
↦ V(1)+a^25
↦ Codeword:
↦ 0,0,0,1,0,1,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ Received word:
↦ 0,0,0,1,1,1,0,1,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ Groebner basis of the QE system:
↦ U(25)+a^7,
↦ U(24)+a^21,
↦ U(23)+a^5,
↦ U(22)+a^9,
↦ U(21)+a^21,
↦ U(20)+a^11,
↦ U(19)+a^22,
↦ U(18)+a^14,
↦ U(17)+a,
↦ U(16)+a^11,
↦ U(15)+a^13,
↦ U(14)+a^10,
↦ U(13)+a^19,
↦ U(12)+a^18,
↦ U(11)+a^26,
↦ U(10)+a^11,
↦ U(9)+a^16,
↦ U(8)+a^22,
↦ U(7)+a^25,
↦ U(6)+a^13,
↦ U(5)+a^8,
↦ U(4)+a^28,
↦ U(3)+a^4,
↦ U(2)+a^2,
↦ U(1),
↦ V(2)+a^2,
↦ V(1)+a^11
```

D.10.2.16 vanishId

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: vanishId (points); point is a list of matrices
'points' is a list of points for which the vanishing ideal is to be constructed

Return: Vanishing ideal corresponding to the given set of points

Example:

```
LIB "decodegb.lib";
ring r=3,(x(1..3)),dp;
//generate all 3-vectors over GF(3)
list points=pointsGen(3,1);
list points2=convPoints(points);
//grasps the first 11 points
list p=graspList(points2,1,11);
print(p);
↦ [1]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=0
↦ [2]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=1
↦ [3]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=-1
↦ [4]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=0
↦ [5]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=1
↦ [6]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=-1
↦ [7]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=0
↦ [8]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=1
↦ [9]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=-1
↦ [10]:
↦    _[1,1]=1
↦    _[2,1]=0
↦    _[3,1]=0
↦ [11]:
↦    _[1,1]=1
↦    _[2,1]=0
↦    _[3,1]=1
//construct the vanishing ideal
ideal id=vanishId(p);
print(id);
↦ x(1)*x(2),
↦ x(1)^2-x(1),
↦ x(3)^3-x(3),
↦ x(1)*x(3)^2-x(1)*x(3),
↦ x(2)^3-x(2)
```

D.10.2.17 sysFL

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: sysFL (check,y,t,e,s); check,y matrix, t,e,s int

- check is a parity check matrix of the code,
- y is a received word,
- t the number of errors to correct,
- e is the extension degree,
- s is the dimension of the point for the vanishing ideal

Return: the system of Fitzgerald-Lax for the given decoding problem

Theory: Based on 'check' of the given linear code, the procedure constructs the corresponding ideal constructed with a generalization of Cooper's philosophy. For basics of the method Section C.8.4 [Fitzgerald-Lax method], page 788.

Example:

```
LIB "decodegb.lib";
intvec vopt = option(get);
list l=FLpreprocess(3,1,11,2,"");
def r=l[1];
setring r;
int s_work=l[2];
//the check matrix of [11,6,5] ternary code
matrix h[5][11]=1,0,0,0,0,1,1,1,-1,-1,0,
0,1,0,0,0,1,1,-1,1,0,-1,
0,0,1,0,0,1,-1,1,0,1,-1,
0,0,0,1,0,1,-1,0,1,-1,1,
0,0,0,0,1,1,0,-1,-1,1,1;
matrix g=dual_code(h);
matrix x[1][6];
matrix y[1][11]=encode(x,g);
//disturb with 2 errors
matrix rec[1][11]=errorInsert(y,list(2,4),list(1,-1));
//the Fitzgerald-Lax system
ideal sys=sysFL(h,rec,2,1,s_work);
print(sys);
↦ x1(3)^3-x1(3),
↦ x1(2)^3-x1(2),
↦ x1(3)^2*x1(1)-x1(3)*x1(1),
↦ x1(2)*x1(1),
↦ x1(1)^2-x1(1),
↦ x1(6)^3-x1(6),
↦ x1(5)^3-x1(5),
↦ x1(6)^2*x1(4)-x1(6)*x1(4),
↦ x1(5)*x1(4),
↦ x1(4)^2-x1(4),
↦ x1(1)^3-x1(1),
↦ x1(4)^3-x1(4),
↦ e(1)^2-1,
↦ e(2)^2-1,
↦ -e(2)*x1(6)^2+e(2)*x1(6)*x1(5)^2-e(2)*x1(6)*x1(4)+e(2)*x1(5)^2+e(2)*x1(5)\
   +e(2)*x1(4)+e(2)-e(1)*x1(3)^2+e(1)*x1(3)*x1(2)^2-e(1)*x1(3)*x1(1)+e(1)*x1\
   (2)^2+e(1)*x1(2)+e(1)*x1(1)+e(1),
↦ -e(2)*x1(6)^2+e(2)*x1(6)*x1(5)^2+e(2)*x1(6)*x1(5)+e(2)*x1(6)*x1(4)-e(2)*x\
   1(6)-e(2)*x1(5)^2+e(2)*x1(5)-e(1)*x1(3)^2+e(1)*x1(3)*x1(2)^2+e(1)*x1(3)*x\
   1(2)+e(1)*x1(3)*x1(1)-e(1)*x1(3)-e(1)*x1(2)^2+e(1)*x1(2)-1,
↦ -e(2)*x1(6)^2*x1(5)^2+e(2)*x1(6)^2*x1(5)-e(2)*x1(6)^2-e(2)*x1(6)*x1(5)^2+\
   e(2)*x1(6)*x1(5)+e(2)*x1(6)*x1(4)+e(2)*x1(6)+e(2)*x1(5)^2-e(2)*x1(5)+e(2)\
   *x1(4)-e(1)*x1(3)^2*x1(2)^2+e(1)*x1(3)^2*x1(2)-e(1)*x1(3)^2-e(1)*x1(3)*x1\
   (2)^2+e(1)*x1(3)*x1(2)+e(1)*x1(3)*x1(1)+e(1)*x1(3)+e(1)*x1(2)^2-e(1)*x1(2\
   )+e(1)*x1(1),
↦ -e(2)*x1(6)^2*x1(5)^2-e(2)*x1(6)^2*x1(5)+e(2)*x1(6)*x1(5)^2-e(2)*x1(6)*x1\
   (4)+e(2)*x1(5)-e(2)*x1(4)-e(1)*x1(3)^2*x1(2)^2-e(1)*x1(3)^2*x1(2)+e(1)*x1\
   (3)*x1(2)^2-e(1)*x1(3)*x1(1)+e(1)*x1(2)-e(1)*x1(1)+1,
↦ e(2)*x1(6)^2*x1(5)+e(2)*x1(4)+e(1)*x1(3)^2*x1(2)+e(1)*x1(1)
option(redSB);
ideal red_sys=std(sys);
red_sys;
↦ red_sys[1]=x1(1)
↦ red_sys[2]=x1(2)^2-x1(2)
↦ red_sys[3]=x1(3)+x1(2)-1
↦ red_sys[4]=e(1)-x1(2)-1
↦ red_sys[5]=x1(4)
↦ red_sys[6]=x1(5)+x1(2)-1
↦ red_sys[7]=x1(6)-x1(2)
↦ red_sys[8]=e(2)+x1(2)+1
// read the solutions from this redGB
// the points are (0,0,1) and (0,1,0) with error values 1 and -1 resp.
// use list points to find error positions;
points;
↦ [1]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=0
↦ [2]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=1
↦ [3]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=-1
↦ [4]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=0
↦ [5]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=1
↦ [6]:
↦    _[1,1]=0
↦    _[2,1]=1
↦    _[3,1]=-1
↦ [7]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=0
↦ [8]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=1
↦ [9]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=-1
↦ [10]:
↦    _[1,1]=1
↦    _[2,1]=0
↦    _[3,1]=0
↦ [11]:
↦    _[1,1]=1
↦      _[2,1]=0
↦      _[3,1]=1
↦   [12]:
↦      _[1,1]=1
↦      _[2,1]=0
↦      _[3,1]=-1
↦   [13]:
↦      _[1,1]=1
↦      _[2,1]=1
↦      _[3,1]=0
↦   [14]:
↦      _[1,1]=1
↦      _[2,1]=1
↦      _[3,1]=1
↦   [15]:
↦      _[1,1]=1
↦      _[2,1]=1
↦      _[3,1]=-1
↦   [16]:
↦      _[1,1]=1
↦      _[2,1]=-1
↦      _[3,1]=0
↦   [17]:
↦      _[1,1]=1
↦      _[2,1]=-1
↦      _[3,1]=1
↦   [18]:
↦      _[1,1]=1
↦      _[2,1]=-1
↦      _[3,1]=-1
↦   [19]:
↦      _[1,1]=-1
↦      _[2,1]=0
↦      _[3,1]=0
↦   [20]:
↦      _[1,1]=-1
↦      _[2,1]=0
↦      _[3,1]=1
↦   [21]:
↦      _[1,1]=-1
↦      _[2,1]=0
↦      _[3,1]=-1
↦   [22]:
↦      _[1,1]=-1
↦      _[2,1]=1
↦      _[3,1]=0
↦   [23]:
↦      _[1,1]=-1
↦      _[2,1]=1
↦      _[3,1]=1
↦   [24]:
↦      _[1,1]=-1
↦      _[2,1]=1
↦      _[3,1]=-1
↦   [25]:
↦      _[1,1]=-1
↦      _[2,1]=-1
↦      _[3,1]=0
↦   [26]:
↦      _[1,1]=-1
↦      _[2,1]=-1
↦      _[3,1]=1
↦   [27]:
↦      _[1,1]=-1
↦      _[2,1]=-1
↦      _[3,1]=-1
option(set,vopt);
```

See also: Section D.10.2.7 [sysQE], page 1946.

D.10.2.18 decodeRandomFL

Procedure from library `decodegb.lib` (see Section D.10.2 [decodegb_lib], page 1937).

Usage: decodeRandomFL(redun,p,e,n,t,ncodes,ntrials,minpol);

- n is length of codes generated,
- redun = redundancy of codes generated,
- p is the characteristic,
- e is the extension degree,
- t is the number of errors to correct,
- ncodes is the number of random codes to be processed,
- ntrials is the number of received vectors per code to be corrected,
- minpol: due to some peculiarities of SINGULAR one needs to provide minimal polynomial for the extension explicitly

Return: nothing

Example:

```
LIB "decodegb.lib";
// correcting one error for one random binary code of length 25,
// redundancy 14; 10 words are processed
decodeRandomFL(25,14,2,1,1,1,10,"");
↦ Codeword:
↦ 1,0,0,1,0,1,0,1,1,1,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0
↦ Received word
↦ 1,0,0,1,0,1,0,1,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0
↦ Groebner basis of the FL system:
↦ x1(1)+1,
↦ x1(2),
↦ x1(3),
↦ x1(4),
↦ x1(5)+1,
↦ e(1)+1
↦ Codeword:
↦ 1,0,0,1,1,1,1,0,0,1,0,1,1,1,0,0,0,1,1,1,1,1,0,0,0
↦ Received word
↦ 1,0,0,1,1,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0
↦ Groebner basis of the FL system:
```

```
↦ x1(1),
↦ x1(2)+1,
↦ x1(3)+1,
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 0,0,1,1,1,0,1,1,1,0,1,1,1,0,0,0,1,0,0,1,0,0,0,0,0
↦ Received word
↦ 0,0,1,1,1,0,1,1,0,0,1,1,1,0,0,0,1,0,0,1,0,0,0,0,0
↦ Groebner basis of the FL system:
↦ x1(1),
↦ x1(2)+1,
↦ x1(3),
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 0,0,0,1,0,1,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ Received word
↦ 0,0,0,1,1,1,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,0,1,0,1
↦ Groebner basis of the FL system:
↦ x1(1),
↦ x1(2),
↦ x1(3)+1,
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 1,1,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,1,0,1,1,0,1,1,1
↦ Received word
↦ 1,1,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,1,0,1,1,0,1,1,1
↦ Groebner basis of the FL system:
↦ x1(1),
↦ x1(2)+1,
↦ x1(3)+1,
↦ x1(4),
↦ x1(5)+1,
↦ e(1)+1
↦ Codeword:
↦ 0,1,1,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0
↦ Received word
↦ 0,1,1,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0
↦ Groebner basis of the FL system:
↦ x1(1)+1,
↦ x1(2),
↦ x1(3)+1,
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 1,0,0,1,0,1,0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,1,1
↦ Received word
↦ 1,0,0,1,0,1,0,1,0,0,1,0,1,0,1,0,0,0,0,0,1,0,0,1,1
↦ Groebner basis of the FL system:
↦ x1(1),
↦ x1(2),
↦ x1(3)+1,
↦ x1(4)+1,
↦ x1(5)+1,
↦ e(1)+1
↦ Codeword:
↦ 0,1,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1
↦ Received word
↦ 0,1,0,1,1,0,1,1,0,0,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1
↦ Groebner basis of the FL system:
↦ x1(1),
↦ x1(2)+1,
↦ x1(3),
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 0,1,0,0,1,0,0,1,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,1
↦ Received word
↦ 0,1,0,0,1,0,0,1,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0
↦ Groebner basis of the FL system:
↦ x1(1)+1,
↦ x1(2)+1,
↦ x1(3),
↦ x1(4),
↦ x1(5),
↦ e(1)+1
↦ Codeword:
↦ 1,0,0,1,1,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,0,1,1
↦ Received word
↦ 1,0,0,1,1,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,0,1,1
↦ Groebner basis of the FL system:
↦ x1(1)+1,
↦ x1(2),
↦ x1(3)+1,
↦ x1(4),
↦ x1(5)+1,
↦ e(1)+1
```

D.11 System and Control theory

D.11.1 Control theory background

Control systems are usually described by differential (or difference) equations, but their properties of interest are most naturally expressed in terms of the system trajectories (the set of all solutions to the equations). This is formalized by the notion of the system *behavior*. On the other hand, the manipulation of linear system equations can be formalized using algebra, more precisely module theory. The relationship between modules and behaviors is very rich and leads to deep results on system structure.

The key to the module-behavior correspondence is a property of some signal spaces that are modules over the ring of differential (or difference) operators, namely, *the injective cogenerator property.* This property makes it possible to translate any statement on the solution spaces that can be expressed in terms of images and kernels, to an equivalent statement on the modules. Thus analytic properties can be identified with algebraic properties, and conversely, the results of manipulating the modules using computer algebra can be re-translated and interpreted using the language of systems theory. This duality (*algebraic analysis*) is widely used in behavioral systems and control theory today.

For instance, a system is **controllable** (a fundamental property for any control system) if and only if the associated module is torsion-free. This concept can be refined by the so-called controllability degrees. The strongest form of controllability (*flatness*) corresponds to a projective (or even free) module.

Controllability means that one can switch from one system trajectory to another without violating the system law (concatenation of trajectories). For one-dimensional systems (ODE) that evolve in time, this is usually interpreted as switching from a given past trajectory to a desired future trajectory. Thus the system can be forced to behave in an arbitrarily prescribed way.

The extreme case opposed to controllability is **autonomy**: autonomous systems evolve independently according to their law, without being influenceable from the outside. Again, the property can be refined in terms of autonomy degrees.

D.11.2 control_lib

Library: control.lib

Purpose: Algebraic analysis tools for System and Control Theory

Authors: Oleksandr Iena, yena@mathematik.uni-kl.de
Markus Becker, mbecker@mathematik.uni-kl.de
Viktor Levandovskyy, levandov@mathematik.uni-kl.de

Support: Forschungsschwerpunkt 'Mathematik und Praxis' (Project of Dr. E. Zerz and V. Levandovskyy), University of Kaiserslautern

Procedures:

D.11.2.1 control

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: control(R); R a module (R is the matrix of the system of equations to be investigated)

Return: list

Purpose: compute the list of all the properties concerning controllability of the system (behavior), represented by the matrix R

Note: the properties and corresponding data like controllability, flatness, dimension of the system, degree of controllability, kernel and image representations, genericity of parameters, obstructions to controllability, annihilator of torsion submodule and left inverse are investigated

Example:

```
LIB "control.lib";
// a WindTunnel example
ring A = (0,a, omega, zeta, k),(D1, delta),dp;
module R;
```

```
R = [D1+a, -k*a*delta, 0, 0],
[0, D1, -1, 0],
[0, omega^2, D1+2*zeta*omega, -omega^2];
R=transpose(R);
view(R);
↦ D1+(a),(-a*k)*delta,0                ,0         ,
↦ 0     ,D1          ,-1               ,0         ,
↦ 0     ,(omega^2)   ,D1+(2*omega*zeta),(-omega^2)
view(control(R));
↦ number of first nonzero Ext:
↦ 
↦ 2
↦ 
↦ controllable, not reflexive, image representation:
↦ 
↦ (a*omega^2*k)*delta                                               ,
↦ (omega^2)*D1+(a*omega^2)                                          ,
↦ (omega^2)*D1^2+(a*omega^2)*D1                                     ,
↦ D1^3+(a+2*omega*zeta)*D1^2+(2*a*omega*zeta+omega^2)*D1+(a*omega^2)
↦ 
↦ dimension of the system:
↦ 
↦ 2
↦ 
↦ Parameter constellations which might lead to a non-controllable system:
↦ 
↦ a,k,omega
↦ 
↦ 
```

D.11.2.2 controlDim

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: controlDim(R); R a module (R is the matrix of the system of equations to be investigated)

Return: list

Purpose: computes list of all the properties concerning controllability of the system (behavior), represented by the matrix R

Note: the properties and corresponding data like controllability, flatness, dimension of the system, degree of controllability, kernel and image representations, genericity of parameters, obstructions to controllability, annihilator of torsion submodule and left inverse are investigated.
This procedure is analogous to 'control' but uses dimension calculations.
The implemented approach works for full row rank matrices only (the check is done automatically).

Example:

```
LIB "control.lib";
//a WindTunnel example
ring A = (0,a, omega, zeta, k),(D1, delta),dp;
module R;
```

```
R = [D1+a, -k*a*delta, 0, 0],
[0, D1, -1, 0],
[0, omega^2, D1+2*zeta*omega, -omega^2];
R=transpose(R);
view(R);
↦ D1+(a),(-a*k)*delta,0                ,0          ,
↦ 0     ,D1          ,-1               ,0          ,
↦ 0     ,(omega^2)   ,D1+(2*omega*zeta),(-omega^2)
view(controlDim(R));
↦ number of first nonzero Ext:
↦
↦ 2
↦
↦ controllable, not reflexive, image representation:
↦
↦ (a*omega^2*k)*delta                                              ,
↦ (omega^2)*D1+(a*omega^2)                                         ,
↦ (omega^2)*D1^2+(a*omega^2)*D1                                    ,
↦ D1^3+(a+2*omega*zeta)*D1^2+(2*a*omega*zeta+omega^2)*D1+(a*omega^2)
↦
↦ dimension of the system:
↦
↦ 2
↦
↦ Parameter constellations which might lead to a non-controllable system:
↦
↦ a,k,omega
↦
↦
```

D.11.2.3 autonom

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: autonom(R); R a module (R is a matrix of the system of equations which is to be investigated)

Return: list

Purpose: find all the properties concerning autonomy of the system (behavior) represented by the matrix R

Note: the properties and corresponding data like autonomy resp. strong autonomy, dimension of the system, autonomy degree, kernel representation and (over)determinacy are investigated

Example:

```
LIB "control.lib";
// Cauchy
ring r=0,(s1,s2,s3,s4),dp;
module R= [s1,-s2],
[s2, s1],
[s3,-s4],
[s4, s3];
R=transpose(R);
```

```
view( R );
↦ s1,-s2,
↦ s2,s1 ,
↦ s3,-s4,
↦ s4,s3
view( autonom(R) );
↦ number of first nonzero Ext:
↦
↦ 2
↦
↦ overdetermined, not strongly autonomous
↦
↦ dimension of the system:
↦
↦ 2
↦
```

D.11.2.4 autonomDim

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: autonomDim(R); R a module (R is a matrix of the system of equations which is to be investigated)

Return: list

Purpose: computes the list of all the properties concerning autonomy of the system (behavior), represented by the matrix R

Note: the properties and corresponding data like autonomy resp. strong autonomy, dimension of the system, autonomy degree, kernel representation and (over)determinacy are investigated.
This procedure is analogous to 'autonom' but uses dimension calculations

Example:

```
LIB "control.lib";
// Cauchy1 example
ring r=0,(s1,s2,s3,s4),dp;
module R= [s1,-s2],
[s2, s1],
[s3,-s4],
[s4, s3];
R=transpose(R);
view( R );
↦ s1,-s2,
↦ s2,s1 ,
↦ s3,-s4,
↦ s4,s3
view( autonomDim(R) );
↦ number of first nonzero Ext:
↦
↦ 2
↦
↦ overdetermined, not strongly autonomous
↦
```

```
↦ dimension of the system:
↦
↦ 2
↦
```

D.11.2.5 leftKernel

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: leftKernel(M); M a matrix

Return: module

Purpose: computes left kernel of matrix M (a module of all elements v such that vM=0)

Example:

```
LIB "control.lib";
ring r= 0,(x,y,z),dp;
matrix M[3][1] = x,y,z;
print(M);
↦ x,
↦ y,
↦ z
matrix L = leftKernel(M);
print(L);
↦ 0, -z,y,
↦ -y,x, 0,
↦ -z,0, x
// check:
print(L*M);
↦ 0,
↦ 0,
↦ 0
```

D.11.2.6 rightKernel

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: rightKernel(M); M a matrix

Return: module

Purpose: computes the right kernel of matrix M (a module of all elements v such that Mv=0)

Example:

```
LIB "control.lib";
ring r = 0,(x,y,z),dp;
matrix M[1][3] = x,y,z;
print(M);
↦ x,y,z
matrix R = rightKernel(M);
print(R);
↦ 0, -y,-z,
↦ -z,x, 0,
↦ y, 0, x
// check:
print(M*R);
↦ 0,0,0
```

D.11.2.7 leftInverse

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: leftInverse(M); M a module

Return: module

Purpose: computes such a matrix L, that LM = Id;

Note: exists only in the case when M is free submodule

Example:

```
LIB "control.lib";
// a trivial example:
ring r = 0,(x,z),dp;
matrix M[2][1] = 1,x2z;
print(M);
↦ 1,
↦ x2z
print( leftInverse(M) );
↦ 1,0
kill r;
// derived from the example TwoPendula:
ring r=(0,m1,m2,M,g,L1,L2),Dt,dp;
matrix U[3][1];
U[1,1]=(-L2)*Dt^4+(g)*Dt^2;
U[2,1]=(-L1)*Dt^4+(g)*Dt^2;
U[3,1]=(L1*L2)*Dt^4+(-g*L1-g*L2)*Dt^2+(g^2);
module M = module(U);
module L = leftInverse(M);
print(L);
↦ (L1^2)/(g^2*L1-g^2*L2),(-L2^2)/(g^2*L1-g^2*L2),1/(g^2)
// check
print(matrix(L)*matrix(M));
↦ 1
```

D.11.2.8 rightInverse

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: rightInverse(M); M a module

Return: module

Purpose: computes such a matrix L, that ML = Id

Note: exists only in the case when M is free submodule

Example:

```
LIB "control.lib";
// a trivial example:
ring r = 0,(x,z),dp;
matrix M[1][2] = 1,x2+z;
print(M);
↦ 1,x2+z
print( rightInverse(M) );
↦ 1,
↦ 0
kill r;
// derived from the TwoPendula example:
ring r=(0,m1,m2,M,g,L1,L2),Dt,dp;
matrix U[1][3];
U[1,1]=(-L2)*Dt^4+(g)*Dt^2;
U[1,2]=(-L1)*Dt^4+(g)*Dt^2;
U[1,3]=(L1*L2)*Dt^4+(-g*L1-g*L2)*Dt^2+(g^2);
module M = module(U);
module L = rightInverse(M);
print(L);
↦ (L1^2)/(g^2*L1-g^2*L2),
↦ (-L2^2)/(g^2*L1-g^2*L2),
↦ 1/(g^2)
// check
print(matrix(M)*matrix(L));
↦ 1
```

D.11.2.9 colrank

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: colrank(M); M a matrix/module

Return: int

Purpose: compute the column rank of M as of matrix

Note: this procedure uses Bareiss algorithm

Example:

```
LIB "control.lib";
// de Rham complex
ring r=0,(D(1..3)),dp;
module R;
R=[0,-D(3),D(2)],
[D(3),0,-D(1)],
[-D(2),D(1),0];
R=transpose(R);
colrank(R);
↦ 2
```

D.11.2.10 genericity

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: genericity(M); M is a matrix/module

Return: list (of strings)

Purpose: determine parametric expressions which have been assumed to be non-zero in the process of computing the Groebner basis

Note: the output list consists of strings. The first string contains the variables only, whereas each further string contains a single polynomial in parameters.
We strongly recommend to switch on the redSB and redTail options.
The procedure is effective with the lift procedure for modules with parameters

Example:

```
LIB "control.lib";
// TwoPendula
ring r=(0,m1,m2,M,g,L1,L2),Dt,dp;
module RR =
[m1*L1*Dt^2, m2*L2*Dt^2, -1, (M+m1+m2)*Dt^2],
[m1*L1^2*Dt^2-m1*L1*g, 0, 0, m1*L1*Dt^2],
[0, m2*L2^2*Dt^2-m2*L2*g, 0, m2*L2*Dt^2];
module R = transpose(RR);
module SR = std(R);
matrix T = lift(R,SR);
genericity(T);
↦ [1]:
↦    m1,g,L1,L2
↦ [2]:
↦    L1-L2
//-- The result might be different when computing reduced bases:
matrix T2;
option(redSB);
option(redTail);
module SR2 = std(R);
T2 =  lift(R,SR2);
genericity(T2);
↦ [1]:
↦    m1,g,L1,m2,L2
↦ [2]:
↦    L1-L2
```

D.11.2.11 canonize

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: canonize(L); L a list

Return: list

Purpose: modules in the list are canonized by computing their reduced minimal (= unique up to constant factor w.r.t. the given ordering) Groebner bases

Assume: L is the output of control/autonomy procedures

Example:

```
LIB "control.lib";
// TwoPendula with L1=L2=L
ring r=(0,m1,m2,M,g,L),Dt,dp;
module RR =
[m1*L*Dt^2, m2*L*Dt^2, -1, (M+m1+m2)*Dt^2],
[m1*L^2*Dt^2-m1*L*g, 0, 0, m1*L*Dt^2],
[0, m2*L^2*Dt^2-m2*L*g, 0, m2*L*Dt^2];
module R = transpose(RR);
list C = control(R);
list CC = canonize(C);
view(CC);
↦ number of first nonzero Ext:
↦
```

```
↦ 1
↦ 
↦ not controllable , image representation for controllable part:
↦ 
↦ -Dt^2                           ,
↦ -Dt^2                           ,
↦ (M*L)*Dt^4+(-m1*g-m2*g-M*g)*Dt^2,
↦ (L)*Dt^2+(-g)
↦ 
↦ kernel representation for controllable part:
↦ 
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
↦ 
↦ obstruction to controllability
↦ 
↦ 1,0,0            ,
↦ 0,1,0            ,
↦ 0,0,(L)*Dt^2+(-g)
↦ 
↦ annihilator of torsion module (of obstruction to controllability)
↦ 
↦ (L)*Dt^2+(-g)
↦ 
↦ dimension of the system:
↦ 
↦ 1
↦ 
```

D.11.2.12 iostruct

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: iostruct(R); R a module

Return: list L with entries: string s, intvec v, module P and module Q

Purpose: if R is the kernel-representation-matrix of some system, then we output a input-ouput representation Py=Qu of the system, the components that have been chosen as outputs(intvec v) and a comment s

Note: the procedure uses Bareiss algorithm

Example:

```
LIB "control.lib";
//Example Antenna
ring r = (0, K1, K2, Te, Kp, Kc),(Dt, delta), (c,dp);
module RR;
RR =
[Dt, -K1, 0, 0, 0, 0, 0, 0, 0],
[0, Dt+K2/Te, 0, 0, 0, 0, -Kp/Te*delta, -Kc/Te*delta, -Kc/Te*delta],
[0, 0, Dt, -K1, 0, 0, 0, 0, 0],
[0, 0, 0, Dt+K2/Te, 0, 0, -Kc/Te*delta, -Kp/Te*delta, -Kc/Te*delta],
[0, 0, 0, 0, Dt, -K1, 0, 0, 0],
[0, 0, 0, 0, 0, Dt+K2/Te, -Kc/Te*delta, -Kc/Te*delta, -Kp/Te*delta];
module R = transpose(RR);
view(iostruct(R));
↦ The following components have been chosen as outputs:
↦
↦ 1,
↦ 2,
↦ 3,
↦ 4,
↦ 5,
↦ 6
↦
↦ Dt,(-K1)       ,0 ,0           ,0 ,0           ,
↦ 0 ,Dt+(K2)/(Te),0 ,0           ,0 ,0           ,
↦ 0 ,0           ,Dt,(-K1)       ,0 ,0           ,
↦ 0 ,0           ,0 ,Dt+(K2)/(Te),0 ,0           ,
↦ 0 ,0           ,0 ,0           ,Dt,(-K1)       ,
↦ 0 ,0           ,0 ,0           ,0 ,Dt+(K2)/(Te)
↦
↦ 0               ,0               ,0               ,
↦ (-Kp)/(Te)*delta,(-Kc)/(Te)*delta,(-Kc)/(Te)*delta,
↦ 0               ,0               ,0               ,
↦ (-Kc)/(Te)*delta,(-Kp)/(Te)*delta,(-Kc)/(Te)*delta,
↦ 0               ,0               ,0               ,
↦ (-Kc)/(Te)*delta,(-Kc)/(Te)*delta,(-Kp)/(Te)*delta
↦
```

D.11.2.13 findTorsion

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: findTorsion(R, I); R an ideal/matrix/module, I an ideal

Return: module

Purpose: computes the Groebner basis of the submodule of R, annihilated by I

Note: especially helpful, when I is the annihilator of the t(R) - the torsion submodule of R. In this case, the result is the explicit presentation of t(R) as the submodule of R

Example:

```
LIB "control.lib";
// Flexible Rod
ring A = 0,(D1, D2), (c,dp);
module R= [D1, -D1*D2, -1], [2*D1*D2, -D1-D1*D2^2, 0];
module RR = transpose(R);
list L = control(RR);
// here, we have the annihilator:
ideal LAnn = D1; // = L[10]
module Tr  = findTorsion(RR,LAnn);
print(RR);  // the module itself
↦ D1,     -D1*D2,     -1,
↦ 2*D1*D2,-D1*D2^2-D1,0
print(Tr); // generators of the torsion submodule
↦ 0,
↦ 1
```

D.11.2.14 controlExample

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: controlExample(s); s a string

Return: ring

Purpose: set up an example from the mini database by initializing a ring and a module in a ring

Note: in order to see the list of available examples, execute `controlExample("show");`
To use an example, one has to do the following. Suppose one calls the ring, where the example will be activated, A. Then, by executing
`def A = controlExample("Antenna");` and `setring A;`,
A will become a basering from the example "Antenna" with the predefined system module R (transposed). After that one can just execute `control(R);` respectively `autonom(R);` to perform the control resp. autonomy analysis of R.

Example:

```
LIB "control.lib";
controlExample("show");   // let us see all available examples:
↦ The list of examples:
↦ name: Cauchy1,  desc: 1-dimensional Cauchy equation
↦ name: Cauchy2,  desc: 2-dimensional Cauchy equation
↦ name: Control1,  desc: example of a simple noncontrollable system
↦ name: Control2,  desc: example of a simple controllable system
↦ name: Antenna,  desc: antenna
↦ name: Einstein,  desc: Einstein equations in vacuum
↦ name: FlexibleRod,  desc: flexible rod
↦ name: TwoPendula,  desc: two pendula mounted on a cart
↦ name: WindTunnel,  desc: wind tunnel
↦ name: Zerz1,  desc: example from the lecture of Eva Zerz
def B = controlExample("TwoPendula"); // let us set up a particular example
setring B;
print(R);
↦ (m1*L1)*Dt^2,             (m2*L2)*Dt^2,             -1,(m1+m2+M)*Dt^2,
↦ (m1*L1^2)*Dt^2+(-m1*g*L1),0,                        0, (m1*L1)*Dt^2,
↦ 0,                        (m2*L2^2)*Dt^2+(-m2*g*L2),0, (m2*L2)*Dt^2
```

D.11.2.15 view

Procedure from library `control.lib` (see Section D.11.2 [control_lib], page 1966).

Usage: view(M); M is of any type

Return: no return value

Purpose: procedure for (well-) formatted output of modules, matrices, lists of modules, matrices; shows everything even if entries are long

Note: in case of other types(not 'module', 'matrix', 'list') works just as standard 'print' procedure

Example:

```
LIB "control.lib";
ring r;
list L;
```

```
matrix M[1][3] = x2+x,y3-y,z5-4z+7;
L[1] = "a matrix:";
L[2] = M;
L[3] = "an ideal:";
L[4] = ideal(M);
view(L);
↦ a matrix:
↦
↦ x2+x,y3-y,z5-4z+7
↦
↦ an ideal:
↦
↦ x2+x,
↦ y3-y,
↦ z5-4z+7
↦
```

D.11.3 jacobson_lib

Library: jacobson.lib

Purpose: Algorithms for Smith and Jacobson Normal Form

Author: Kristina Schindelar, Kristina.Schindelar@math.rwth-aachen.de,
Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview: We work over a ring R, that is an Euclidean principal ideal domain. If R is commutative, we suppose R to be a polynomial ring in one variable. If R is non-commutative, we suppose R to have two variables, say x and d. We treat then the basering as the Ore localization of R with respect to the mult. closed set S = K[x] without 0. Thus, we treat basering as principal ideal ring with d a polynomial variable and x an invertible one.
Note, that in computations no division by x will actually happen.

Given a rectangular matrix M over R, one can compute unimodular (that is invertible) square matrices U and V, such that U*M*V=D is a diagonal matrix. Depending on the ring, the diagonal entries of D have certain properties.

We call a square matrix D as before 'a weak Jacobson normal form of M'. It is known, that over the first rational Weyl algebra K(x)<d>, D can be further transformed into a diagonal matrix (1,1,...,1,f,0,..,0), where f is in K(x)<d>. We call such a form of D the strong Jacobson normal form. The existence of strong form in not guaranteed if one works with algebra, which is not rational Weyl algebra.

References:

[1] N. Jacobson, 'The theory of rings', AMS, 1943.
[2] Manuel Avelino Insua Hermo, 'Varias perspectives sobre las bases de Groebner : Forma normal de Smith, Algorithme de Berlekamp y algebras de Leibniz'.
PhD thesis, Universidad de Santiago de Compostela, 2005.
[3] V. Levandovskyy, K. Schindelar 'Computing Jacobson normal form using Groebner bases',
to appear in Journal of Symbolic Computation, 2010.

Procedures: See also: Section D.11.2 [control_lib], page 1966.

D.11.3.1 smith

Procedure from library `jacobson.lib` (see Section D.11.3 [jacobson_lib], page 1977).

Usage: smith(M[, eng1, eng2]); M matrix, eng1 and eng2 are optional integers

Return: matrix or list of matrices, depending on arguments

Assume: Basering is a commutative polynomial ring in one variable

Purpose: compute the Smith Normal Form of M with (optionally) transformation matrices

Theory: Groebner bases are used for the Smith form like in [2] and [3].

Note: By default, just the Smith normal form of M is returned.
If the optional integer `eng1` is non-zero, the list {U,D,V} is returned
where U*M*V = D and the diagonal field entries of D are not normalized.
The normalization of the latter can be done with the 'divideUnits' procedure.
U and V above are square unimodular (invertible) matrices.
Note, that the procedure works for a rectangular matrix M.

The optional integer `eng2` determines the Groebner basis engine:
0 (default) ensures the use of 'slimgb' , otherwise 'std' is used.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "jacobson.lib";
ring r = 0,x,Dp;
matrix m[3][2]=x, x^4+x^2+21, x^4+x^2+x, x^3+x, 4*x^2+x, x;
list s=smith(m,1);
print(s[2]);  // non-normalized Smith form of m
↦ 21,0,
↦ 0, x,
↦ 0, 0
print(s[1]*m*s[3] - s[2]); // check U*M*V = D
↦ 0,0,
↦ 0,0,
↦ 0,0
list t = divideUnits(s);
print(t[2]); // the Smith form of m
↦ 1,0,
↦ 0,x,
↦ 0,0
```

See also: Section D.11.3.3 [divideUnits], page 1979; Section D.11.3.2 [jacobson], page 1978.

D.11.3.2 jacobson

Procedure from library `jacobson.lib` (see Section D.11.3 [jacobson_lib], page 1977).

Usage: jacobson(M, eng); M matrix, eng an optional int

Return: list

Assume: Basering is a (non-commutative) ring in two variables.

Purpose: compute a weak Jacobson normal form of M over the basering

Theory: Groebner bases and involutions are used, following [3]

Note: A list L of matrices {U,D,V} is returned. That is L[1]*M*L[3]=L[2],
where L[2] is a diagonal matrix and
L[1], L[3] are square invertible polynomial (unimodular) matrices.
Note, that M can be rectangular.
The optional integer `eng2` determines the Groebner basis engine:
0 (default) ensures the use of 'slimgb' , otherwise 'std' is used.

Display: If `printlevel`=1, progress debug messages will be printed,
if `printlevel`>=2, all the debug messages will be printed.

Example:

```
LIB "jacobson.lib";
ring r = 0,(x,d),Dp;
def R = nc_algebra(1,1);   setring R; // the 1st Weyl algebra
matrix m[2][2] = d,x,0,d; print(m);
↦ d,x,
↦ 0,d
list J = jacobson(m); // returns a list with 3 entries
print(J[2]); // a Jacobson Form D for m
↦ xd2-d,0,
↦ 0,    1
print(J[1]*m*J[3] - J[2]); // check that U*M*V = D
↦ 0,0,
↦ 0,0
/*   now, let us do the same for the shift algebra  */
ring r2 = 0,(x,s),Dp;
def R2 = nc_algebra(1,s);   setring R2; // the 1st shift algebra
matrix m[2][2] = s,x,0,s; print(m); // matrix of the same for as above
↦ s,x,
↦ 0,s
list J = jacobson(m);
print(J[2]); // a Jacobson Form D, quite different from above
↦ xs2+s2,0,
↦ 0,     x
print(J[1]*m*J[3] - J[2]); // check that U*M*V = D
↦ 0,0,
↦ 0,0
```

See also: Section D.11.3.3 [divideUnits], page 1979; Section D.11.3.1 [smith], page 1978.

D.11.3.3 divideUnits

Procedure from library `jacobson.lib` (see Section D.11.3 [jacobson_lib], page 1977).

Usage: divideUnits(L); list L

Return: matrix or list of matrices

Assume: L is an output of `smith` or a `jacobson` procedures, that is
either L contains one rectangular matrix with elements only on the main diagonal
or L consists of three matrices, where L[1] and L[3] are square invertible matrices
while L[2] is a rectangular matrix with elements only on the main diagonal

Purpose: divide out units from the diagonal and reflect this in transformation matrices

Example:

```
LIB "jacobson.lib";
ring R=(0,m,M,L1,L2,m1,m2,g), D, lp; // two pendula example
matrix P[3][4]=m1*L1*D^2,m2*L2*D^2,(M+m1+m2)*D^2,-1,
m1*L1^2*D^2-m1*L1*g,0,m1*L1*D^2,0,0,
m2*L2^2*D^2-m2*L2*g,m2*L2*D^2,0;
list s=smith(P,1);  // returns a list with 3 entries
print(s[2]); // a diagonal form, close to the Smith form
↦ (L1*L2*m2*g^2-L2^2*m2*g^2),0,   0,    0,
↦ 0,                         (L2),0,    0,
↦ 0,                         0,   (g^2),0
print(s[1]); // U, left transformation matrix
↦ 0,    (-L2*m2)/(L1*m1),    1,
↦ (-L2),(M*L2+L2*m1)/(L1*m1),1,
↦ 0,    1/(L1*m1),           0
list t = divideUnits(s);
print(t[2]); // the Smith form of the matrix P
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,0,1,0
print(t[1]); // U', modified left transformation matrix
↦ 0, -1/(L1^2*m1*g^2-L1*L2*m1*g^2),1/(L1*L2*m2*g^2-L2^2*m2*g^2),
↦ -1,(M+m1)/(L1*m1),               1/(L2),
↦ 0, 1/(L1*m1*g^2),                0
```

D.11.4 findifs_lib

Library: findifs.lib

Purpose: Tools for the finite difference schemes

Authors: Viktor Levandovskyy, levandov@math.rwth-aachen.de

Overview: We provide the presentation of difference operators in a polynomial, semi-factorized and a nodal form. Running `findifs_example();` will demonstrate, how we generate finite difference schemes of linear PDEs from given approximations.

Theory: The method we use have been developed by V. Levandovskyy and Bernd Martin. The computation of a finite difference scheme of a given single linear partial differential equation with constant coefficients with a given approximation rules boils down to the computation of a Groebner basis of a submodule of a free module with respect to the ordering, eliminating module components.

Support: SpezialForschungsBereich F1301 of the Austrian FWF

Procedures: See also: Section D.15.4 [finitediff_lib], page 2329; Section D.9.2 [latex_lib], page 1906.

D.11.4.1 findifs_example

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: findifs_example();

Return: nothing (demo)

Purpose: demonstration of our approach and this library

Example:

```
LIB "findifs.lib";
findifs_example();
↦ * Equation: u_tt - A^2 u_xx -B^2 u_yy = 0; A,B are constants
↦ * we employ three central differences
↦ * the vector we act on is (u_xx, u_yy, u_tt, u)^T
↦ * Set up the ring:
↦ ring r = (0,A,B,dt,dx,dy),(Tx,Ty,Tt),(c,dp);
↦ * Set up the matrix with equation and approximations:
↦ matrix M[4][4] =
↦     // direct equation:
↦     -A^2, -B^2, 1, 0,
↦     // central difference u_tt
↦     0, 0,  -dt^2*Tt, (Tt-1)^2,
↦     // central difference u_xx
↦     -dx^2*Tx, 0, 0, (Tx-1)^2,
↦     // central difference u_yy
↦     0, -dy^2*Ty, 0, (Ty-1)^2;
↦ * Print the differential form of equations:
↦ (-A^2)*Uxx+(-B^2)*Uyy+Utt,
↦ (-dt^2*Tt)*Utt+(Tt^2-2*Tt+1)*U,
↦ (-dx^2*Tx)*Uxx+(Tx^2-2*Tx+1)*U,
↦ (-dy^2*Ty)*Uyy+(Ty^2-2*Ty+1)*U
↦ * Perform the elimination of module components:
↦  module R = transpose(M);
↦  module S = std(R);
↦  * See the result of Groebner bases: generators are columns
↦  print(S);
↦ 0,     0,           0,     0,           0,         (A^2),
↦ 0,     0,           0,     (dy^2)*Ty,   S[2,5],    (B^2),
↦ 0,     (dt^2)*Tt,   S[3,3],0,           (-dx^2)*Tx,-1,
↦ S[4,1],-Tt^2+2*Tt-1,S[4,3],-Ty^2+2*Ty-1,S[4,5],    0
↦  * So, only the first column has its nonzero element in the last componen\
  t
↦  * Hence, this polynomial is the scheme
↦  poly   p = S[4,1];
↦  print(p);
↦ (A^2*dt^2*dy^2)*Tx^2*Ty*Tt+(B^2*dt^2*dx^2)*Tx*Ty^2*Tt+(-dx^2*dy^2)*Tx*Ty*\
  Tt^2+(-2*A^2*dt^2*dy^2-2*B^2*dt^2*dx^2+2*dx^2*dy^2)*Tx*Ty*Tt+(-dx^2*dy^2)\
  *Tx*Ty+(B^2*dt^2*dx^2)*Tx*Tt+(A^2*dt^2*dy^2)*Ty*Tt
↦ * Create the nodal of the scheme in TeX format:
↦  ideal I = decoef(p,dt);
↦  difpoly2tex(I,L);
↦ \frac{1}{\tri t^{2}}\cdot (u^{n+2}_{j+1,k+1}+(-2) u^{n+1}_{j+1,k+1}+u^{n}\
  _{j+1,k+1})+ \frac{-\lambda^{2}}{\tri x^{2}}\cdot (u^{n+1}_{j+2,k+1}+\fra\
  c{B^{2}\tri x^{2}}{\lambda^{2}\tri y^{2}} u^{n+1}_{j+1,k+2}+\frac{-(2\lam\
  bda^{2}\tri y^{2}+2B^{2}\tri x^{2}}{\lambda^{2}\tri y^{2}} u^{n+1}_{j+1,k\
  +1}+\frac{B^{2}\tri x^{2}}{\lambda^{2}\tri y^{2}} u^{n+1}_{j+1}+u^{n+1}_{\
  j,k+1})
↦ * Preparations for the semi-factorized form:
↦  poly pi1 = subst(I[2],B,0);
↦  poly pi2 = I[2] - pi1;
↦ * Show the semi-factorized form of the scheme: 1st summand
↦  factorize(I[1]);
↦ [1]:
↦    _[1]=(-dx^2*dy^2)
↦    _[2]=Tx
↦    _[3]=Ty
↦    _[4]=Tt-1
↦ [2]:
↦    1,1,1,2
↦ * Show the semi-factorized form of the scheme: 2nd summand
↦  factorize(pi1);
↦ [1]:
↦    _[1]=(A^2*dt^2*dy^2)
↦    _[2]=Ty
↦    _[3]=Tt
↦    _[4]=Tx-1
↦ [2]:
↦    1,1,1,2
↦ * Show the semi-factorized form of the scheme: 3rd summand
↦  factorize(pi1);
↦ [1]:
↦    _[1]=(B^2*dt^2*dx^2)
↦    _[2]=Tx
↦    _[3]=Tt
↦    _[4]=Ty-1
↦ [2]:
↦    1,1,1,2
```

D.11.4.2 decoef

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: decoef(P,n); P a poly, n a number

Return: ideal

Purpose: decompose polynomial P into summands with respect
to the presence of the number n in the coefficients

Note: n is usually a parameter with no power

Example:

```
LIB "findifs.lib";
" EXAMPLE:";
↦  EXAMPLE:
ring r = (0,dh,dt),(Tx,Tt),dp;
poly P = (4*dh^2-dt)*Tx^3*Tt + dt*dh*Tt^2 + dh*Tt;
decoef(P,dt);
↦ _[1]=(4*dh^2)*Tx^3*Tt+(dh)*Tt
↦ _[2]=(-dt)*Tx^3*Tt+(dh*dt)*Tt^2
decoef(P,dh);
↦ _[1]=(-dt)*Tx^3*Tt
↦ _[2]=(4*dh^2)*Tx^3*Tt+(dh*dt)*Tt^2+(dh)*Tt
```

D.11.4.3 difpoly2tex

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: difpoly2tex(S,P[,Q]); S an ideal, P and optional Q are lists

Return: string

Purpose: present the difference scheme in the nodal form

Assume: ideal S is the result of `decoef` procedure

Note: a list P may be empty or may contain parameters, which will not appear in denominators
an optional list Q represents the part of the scheme, depending on other function, than the major part

Example:

```
LIB "findifs.lib";
ring r = (0,dh,dt,V),(Tx,Tt),dp;
poly M = (4*dh*Tx+dt)^2*(Tt-1) + V*Tt*Tx;
ideal I = decoef(M,dt);
list L; L[1] = V;
difpoly2tex(I,L);
↦ \frac{1}{8\tri t}\cdot (u^{n+1}_{j+2}-u^{n}_{j+2}+\frac{\nu}{16\tri h^{2}\
   } u^{n+1}_{j+1})+ \frac{1}{16\tri h}\cdot (u^{n+1}_{j+1}-u^{n}_{j+1}+\fra\
   c{\tri t}{8\tri h} u^{n+1}_{j}+\frac{-\tri t}{8\tri h} u^{n}_{j})
poly G = V*dh^2*(Tt-Tx)^2;
difpoly2tex(I,L,G);
↦ \frac{1}{8\tri t}\cdot (u^{n+1}_{j+2}-u^{n}_{j+2}+\frac{\nu}{16\tri h^{2}\
   } u^{n+1}_{j+1})+ \frac{1}{16\tri h}\cdot (u^{n+1}_{j+1}-u^{n}_{j+1}+\fra\
   c{\tri t}{8\tri h} u^{n+1}_{j}+\frac{-\tri t}{8\tri h} u^{n}_{j})+ \frac{\
   \nu}{128\tri t}\cdot (p^{n}_{j+2}+(-2) p^{n+1}_{j+1}+p^{n+2}_{j})
```

D.11.4.4 exp2pt

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: exp2pt(P[,L]); P poly, L an optional list of strings

Return: string

Purpose: convert a polynomial M into the TeX format, in nodal form

Assume: coefficients must not be fractional

Note: an optional list L contains a string, which will replace the default value 'u' for the discretized function

Example:

```
LIB "findifs.lib";
" EXAMPLE:";
↦  EXAMPLE:
ring r = (0,dh,dt),(Tx,Tt),dp;
poly M = (4*dh*Tx^2+1)*(Tt-1)^2;
print(exp2pt(M));
↦ 4\tri h u^{n+2}_{j+2}+(-8\tri h) u^{n+1}_{j+2}+4\tri h u^{n}_{j+2}+u^{n+2\
   }_{j}+(-2) u^{n+1}_{j}+u^{n}_{j}
print(exp2pt(M,"F"));
↦ 4\tri h F^{n+2}_{j+2}+(-8\tri h) F^{n+1}_{j+2}+4\tri h F^{n}_{j+2}+F^{n+2\
   }_{j}+(-2) F^{n+1}_{j}+F^{n}_{j}
```

D.11.4.5 texcoef

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: texcoef(n); n a number

Return: string

Purpose: converts the number n into TeX format

Note: if n is a polynomial, texcoef adds extra brackets and performs some space substitutions

Example:

```
LIB "findifs.lib";
" EXAMPLE:";
↦  EXAMPLE:
ring r = (0,dh,dt),(Tx,Tt),dp;
number n1,n2,n3 = dt/(4*dh^2-dt),(dt+dh)^2, 1/dh;
n1; texcoef(n1);
↦ (dt)/(4*dh^2-dt)
↦ \frac{\tri t}{4\tri h^{2}-\tri t}
n2; texcoef(n2);
↦ (dh^2+2*dh*dt+dt^2)
↦ (\tri h^{2}+2\tri h\tri t+\tri t^{2})
n3; texcoef(n3);
↦ 1/(dh)
↦ \frac{1}{\tri h}
```

D.11.4.6 npar

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: npar(n); n a number

Return: int

Purpose: searches for 'n' among the parameters and returns its number

Example:

```
LIB "findifs.lib";
ring r = (0,dh,dt,theta,A),t,dp;
npar(dh);
↦ 1
number T = theta;
npar(T);
↦ 3
npar(dh^2);
↦ Incorrect parameter
↦ 0
```

D.11.4.7 magnitude

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: magnitude(P); P a poly

Return: poly

Purpose: compute the square of the magnitude of a complex expression

Assume: i is the variable of a basering

Example:

```
LIB "findifs.lib";
ring r = (0,d),(g,i,sin,cos),dp;
poly P = d*i*sin - g*cos +d^2*i;
NF( magnitude(P), std(i^2+1) );
↦ g^2*cos^2+(d^2)*sin^2+(2*d^3)*sin+(d^4)
```

D.11.4.8 replace

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: replace(s,what,with); s,what,with strings

Return: string

Purpose: replaces in 's' all the substrings 'what' with substring 'with'

Note:

Example:

```
LIB "findifs.lib";
" EXAMPLE:";
↦  EXAMPLE:
ring r = (0,dt,theta),Tt,dp;
poly p = (Tt*dt+theta+1)^2+2;
string s = texfactorize("",p);
s;
↦ $(dt^{2}\cdot Tt^{2}+(2dttheta+2dt)\cdot Tt+(theta^{2}+2theta+3))$
s = replace(s,"Tt","T_t"); s;
↦ $(dt^{2}\cdot T_t^{2}+(2dttheta+2dt)\cdot T_t+(theta^{2}+2theta+3))$
s = replace(s,"dt","\\tri t"); s;
↦ $(\tri t^{2}\cdot T_t^{2}+(2\tri ttheta+2\tri t)\cdot T_t+(theta^{2}+2the\
   ta+3))$
s = replace(s,"theta","\\theta"); s;
↦ $(\tri t^{2}\cdot T_t^{2}+(2\tri t\theta+2\tri t)\cdot T_t+(\theta^{2}+2\\
   theta+3))$
```

D.11.4.9 xchange

Procedure from library `findifs.lib` (see Section D.11.4 [findifs_lib], page 1980).

Usage: xchange(w,a,b); w,a,b strings

Return: string

Purpose: exchanges substring 'a' with a substring 'b' in the string w

Note:

Example:

```
LIB "findifs.lib";
" EXAMPLE:";
↦  EXAMPLE:
ring r = (0,dt,dh,A),Tt,dp;
poly p = (Tt*dt+dh+1)^2+2*A;
string s = texpoly("",p);
```

```
s;
↦ $dt^{2}\cdot Tt^{2}+(2dtdh+2dt)\cdot Tt+(dh^{2}+2dh+2A+1)$
string t = xchange(s,"dh","dt");
t;
↦ $dh^{2}\cdot Tt^{2}+(2dhdt+2dh)\cdot Tt+(dt^{2}+2dt+2A+1)$
```

D.12 Teaching

The libraries in this section are intended to be used for teaching purposes but not for serious computations.

D.12.1 aksaka_lib

Library: aksaka.lib

Purpose: Procedures for primality testing after Agrawal, Saxena, Kayal

Authors: Christoph Mang

Overview: Algorithms for primality testing in polynomial time based on the ideas of Agrawal, Saxena and Kayal.

Procedures:

D.12.1.1 fastExpt

Procedure from library **aksaka.lib** (see Section D.12.1 [aksaka_lib], page 1986).

Usage: fastExpt(a,m,n); a, m, n = number;

Return: the m-th power of a; if a^m>n the procedure returns n+1

Note: uses fast exponentiation

Example:

```
LIB "aksaka.lib";
fastExpt(2,10,1022);
↦ 1023
```

D.12.1.2 log2

Procedure from library **aksaka.lib** (see Section D.12.1 [aksaka_lib], page 1986).

Usage: log2(x);

Return: logarithm to basis 2 of x

Note: calculates the natural logarithm of x with a power-series of the ln, then the basis is changed to 2

Example:

```
LIB "aksaka.lib";
log2(1024);
↦ 10
```

D.12.1.3 PerfectPowerTest

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: PerfectPowerTest(n);

Return: 0 if there are numbers a,b>1 with a^b=n;
1 if there are no numbers a,b>1 with a^b=n;
if printlevel>=1 and there are a,b>1 with a^b=n,
then a,b are printed

Example:

```
LIB "aksaka.lib";
PerfectPowerTest(887503681);
↦ 0
```

D.12.1.4 wurzel

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: wurzel(r);

Assume: characteristic of basering is 0, r>=0

Return: number, square root of r

Example:

```
LIB "aksaka.lib";
ring R = 0,x,dp;
wurzel(7629412809180100);
↦ 34416277282534323424601232988975951646615355075494301198215038753739930001\
   950914732024064729050404203240063089015829737039655/394020061963944792122\
   7904010014361380507973927046544666794829340424572177149721061141426625488\
   4915640806627990306816
```

D.12.1.5 euler

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: euler(r);

Return: bigint phi(r), where phi is Eulers phi-function

Note: first r is factorized with proc PollardRho, then phi(r) is calculated with the help of phi(p) of every factor p;

Example:

```
LIB "aksaka.lib";
euler(99991);
↦ 99990
```

D.12.1.6 coeffmod

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: coeffmod(f,n);

Assume: poly f depends on at most var(1) of the basering

Return: poly f modulo number n

Note: at first the coefficients of the monomials of the polynomial f are determined, then their remainder modulo n is calculated, after that the polynomial 'is put together' again

Example:

```
LIB "aksaka.lib";
ring R = 0,x,dp;
poly f=2457*x4+52345*x3-98*x2+5;
bigint n=3;
coeffmod(f,n);
↦ x3+x2+2
```

D.12.1.7 powerpolyX

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: powerpolyX(q,n,a,r);

Return: the q-th power of poly a modulo poly r and number n

Example:

```
LIB "aksaka.lib";
ring R=0,x,dp;
poly a=3*x3-x2+5;
poly r=x7-1;
bigint q=123;
bigint n=5;
powerpolyX(q,n,a,r);
↦ 3x5+2x4+x3+x2+1
```

D.12.1.8 ask

Procedure from library `aksaka.lib` (see Section D.12.1 [aksaka_lib], page 1986).

Usage: ask(n);

Assume: n>1

Return: 0 if n is composite;
1 if n is prime;
if printlevel>=1, you are informed what the procedure will do or has calculated

Note: ASK-algorithm; uses proc powerpolyX for step 5

Example:

```
LIB "aksaka.lib";
//ask(100003);
ask(32003);
↦ 1
```

D.12.2 crypto_lib

Library: crypto.lib

Purpose: Procedures for teaching cryptography

Authors: Gerhard Pfister, pfister@mathematik.uni-kl.de
David Brittinger, dativ@gmx.net

Overview: The library contains procedures to compute the discrete logarithm, primality-tests, factorization included elliptic curves. The library is intended to be used for teaching purposes but not for serious computations. Sufficiently high printlevel allows to control each step, thus illustrating the algorithms at work.

Procedures:

D.12.2.1 round

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: round(r);

Return: the nearest number to r out of Z

Assume: r should be a rational or a real number

Example:

```
LIB "crypto.lib";
ring R = (real,50),x,dp;
number r=7357683445788723456321.6788643224;
round(r);
↦ 7357683445788723456322
```

D.12.2.2 bubblesort

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: bubblesort(L);

Return: list L, sort in decreasing order

Example:

```
LIB "crypto.lib";
ring r = 0,x,dp;
list L=-567,-233,446,12,-34,8907;
bubblesort(L);
↦ [1]:
↦    8907
↦ [2]:
↦    446
↦ [3]:
↦    12
↦ [4]:
↦    -34
↦ [5]:
↦    -233
↦ [6]:
↦    -567
```

D.12.2.3 decimal

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: decimal(s); s = string

Return: the (decimal) number corresponding to the hexadecimal number s

Example:

```
LIB "crypto.lib";
string s  ="8edfe37dae96cfd2466d77d3884d4196";
decimal(s);
↦ 189912871665444375716340628395668619670
```

D.12.2.4 eexgcdN

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: eexgcdN(L);

Return: list T such that sum_i L[i]*T[i]=T[n+1]=gcd(L[1],...,L[n])

Example:

```
LIB "crypto.lib";
eexgcdN(list(24,15,21));
↦ [1]:
↦    2
↦ [2]:
↦    -3
↦ [3]:
↦    0
↦ [4]:
↦    3
```

D.12.2.5 lcmN

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: lcmN(a,b);

Return: lcm(a,b);

Example:

```
LIB "crypto.lib";
lcmN(24,15);
↦ 120
```

D.12.2.6 powerN

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: powerN(m,d,n);

Return: m^d mod n

Example:

```
LIB "crypto.lib";
powerN(24,15,7);
↦ 6
```

D.12.2.7 chineseRem

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: chineseRem(T,L);

Return: x such that x = T[i] mod L[i]

Note: chinese remainder theorem

Example:

```
LIB "crypto.lib";
chineseRem(list(24,15,7),list(2,3,5));
↦ 12
```

D.12.2.8 Jacobi

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: Jacobi(a,n);

Return: the generalized Legendre symbol

Note: if n is an odd prime then Jacobi(a,n)=0,1,-1 if n|a, a=x^2 mod n,else

Example:

```
LIB "crypto.lib";
Jacobi(13580555397810650806,5792543);
↦ 1
```

D.12.2.9 primList

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: primList(n);

Return: the list of all primes <=n

Example:

```
LIB "crypto.lib";
list L=primList(100);
size(L);
↦ 25
L[size(L)];
↦ 97
```

D.12.2.10 primL

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: primL(q);

Return: list of the first primes p_1,...,p_r such that q>p_1*...*p_(r-1) and q<p_1*...*p_r

Example:

```
LIB "crypto.lib";
primL(20);
↦ [1]:
↦    2
↦ [2]:
↦    3
↦ [3]:
↦    5
```

D.12.2.11 intPart

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: intPart(x);

Return: the integral part of a rational number

Example:

```
LIB "crypto.lib";
ring r=0,x,dp;
intPart(7/3);
↦ 2
```

D.12.2.12 intRoot

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: intRoot(m);

Return: the integral part of the square root of m

Example:

```
LIB "crypto.lib";
intRoot(20);
↦ 4
```

D.12.2.13 squareRoot

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: squareRoot(a,p);

Return: the square root of a in Z/p, p prime

Note: assumes the Jacobi symbol is 1 or p=2.

Example:

```
LIB "crypto.lib";
squareRoot(8315890421938608,32003);
↦ 18784
```

D.12.2.14 solutionsMod2

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: solutionsMod2(M);

Return: an intmat containing a basis of the vector space of solutions of the linear system of equations defined by M over the prime field of characteristic 2

Example:

```
LIB "crypto.lib";
bigintmat M[3][3]=1,2,3,4,5,6,7,6,5;
solutionsMod2(M);
↦ 1,0,1
```

D.12.2.15 powerX

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: powerX(q,i,I);

Return: the q-th power of the i-th variable modulo I

Assume: I is a standard basis

Example:

```
LIB "crypto.lib";
ring R = 0,(x,y),dp;
powerX(100,2,std(ideal(x3-1,y2-x)));
↦ x2
```

D.12.2.16 babyGiant

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: babyGiant(b,y,p);

Return: the discrete logarithm x: b^x=y mod p

Note: This procedure works based on Shank's baby step - giant step method.

Example:

```
LIB "crypto.lib";
bigint b=2;
bigint y=10;
bigint p=101;
babyGiant(b,y,p);
↦ 25
```

D.12.2.17 rho

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: rho(b,y,p);

Return: the discrete logarithm x=log_b(y): b^x=y mod p

Note: Pollard's rho:
choose random f_0 in 0,...,p-2 ,e_0=0, define x_0=b^f_0, define x_i=y^e_i*b^f_i as below. For i large enough there is i with x_(i/2)=x_i. Let s:=e_(i/2)-e_i mod p-1 and t:=f_i-f_(i/2) mod p-1, d=gcd(s,p-1)=u*s+v*(p-1) then x=tu/d +j*(p-1)/d for some j (to be found by trying)

Example:

```
LIB "crypto.lib";
bigint b=2;
bigint y=10;
bigint p=101;
rho(b,y,p);
↦ 25
```

D.12.2.18 MillerRabin

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: MillerRabin(n,k);

Return: 1 if n is prime, 0 else

Note: probabilistic test of Miller-Rabin with k loops to test if n is prime. Using the theorem: If n is prime, n-1=2^s*r, r odd, then powerN(a,r,n)=1 or powerN(a,r*2^i,n)=-1 for some i

Example:

```
LIB "crypto.lib";
bigint x=2;
x=x^787-1;
MillerRabin(x,3);
↦ 0
```

D.12.2.19 SolowayStrassen

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: SolowayStrassen(n,k);

Return: 1 if n is prime, 0 else

Note: probabilistic test of Soloway-Strassen with k loops to test if n is prime using the theorem: If n is prime then
powerN(a,(n-1)/2,n)=Jacobi(a,n) mod n

Example:

```
LIB "crypto.lib";
bigint h=10;
bigint p=h^100+267;
//p=h^100+43723;
//p=h^200+632347;
SolowayStrassen(h,3);
↦ 0
```

D.12.2.20 PocklingtonLehmer

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: PocklingtonLehmer(N); optional: PocklingtonLehmer(N,L); L a list of the first k primes

Return: message N is not prime or `{A,{p},{a_p}}` as certificate for N being prime

Note: assumes that it is possible to factorize N-1=A*B such that gcd(A,B)=1, the factorization of A is completely known and A^2>N .
N is prime if and only if for each prime factor p of A we can find a_p such that a_p^(N-1)=1 mod N and gcd(a_p^((N-1)/p)-1,N)=1

Example:

```
LIB "crypto.lib";
bigint N=105554676553297;
PocklingtonLehmer(N);
↦ [1]:
```

```
↦    6442503168
↦ [2]:
↦    [1]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          2
↦    [3]:
↦       [1]:
↦          2097169
↦       [2]:
↦          2
list L=primList(1000);
PocklingtonLehmer(N,L);
↦ [1]:
↦    3221246976
↦ [2]:
↦    [1]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          2
↦    [3]:
↦       [1]:
↦          1048583
↦       [2]:
↦          2
```

D.12.2.21 PollardRho

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: PollardRho(n,k,allFactors); optional: PollardRho(n,k,allFactors,L); L a list of the first k primes

Return: a list of factors of n (which could be just n),if allFactors=0
a list of all factors of n ,if allFactors=1

Note: probabilistic rho-algorithm of Pollard to find a factor of n in k loops. Creates a sequence x_i such that (x_i)^2=(x_2i)^2 mod n for some i, computes gcd(x_i-x_2i,n) to find a divisor. To define the sequence choose x,a and define x_n+1=x_n^2+a mod n, x_1=x. If allFactors is 1, it tries to find recursively all prime factors using the Soloway-Strassen test.

Example:

```
LIB "crypto.lib";
bigint h=10;
bigint p=h^30+4;
PollardRho(p,5000,0);
↦ [1]:
↦    2
↦ [2]:
↦    157
↦ [3]:
↦    18737561
↦ [4]:
↦    84982068258408294013
```

See also: Section 5.1.119 [primefactors], page 241.

D.12.2.22 pFactor

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: pFactor(n,B,P); n to be factorized, B a bound , P a list of primes

Return: a list of factors of n or n if no factor found

Note: Pollard's p-factorization
creates the product k of powers of primes (bounded by B) from the list P with the idea that for a prime divisor p of n we have p-1|k, and then p divides gcd(a^k-1,n) for some random a

Example:

```
LIB "crypto.lib";
list L=primList(1000);
pFactor(1241143,13,L);
↦ 547
bigint h=10;
h=h^30+25;
pFactor(h,20,L);
↦ 325
```

D.12.2.23 quadraticSieve

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: quadraticSieve(n,c,B,k); n to be factorized, [-c,c] the sieve-intervall, B a list of primes, k for using the first k elements in B

Return: a list of factors of n or the message: no divisor found

Note: The idea being used is to find x,y such that x^2=y^2 mod n then gcd(x-y,n) can be a proper divisor of n

Example:

```
LIB "crypto.lib";
list L=primList(5000);
quadraticSieve(7429,3,L,4);
↦ 17
quadraticSieve(1241143,100,L,50);
↦ 547
```

D.12.2.24 isOnCurve

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: isOnCurve(N,a,b,P);

Return: 1 or 0 (depending on whether P is on the curve or not)

Note: checks whether P=(P[1]:P[2]:P[3]) is a point on the elliptic curve defined by y^2z=x^3+a*xz^2+b*z^3 over Z/N

Example:

```
LIB "crypto.lib";
isOnCurve(32003,5,7,list(10,16,1));
↦ 0
```

D.12.2.25 ellipticAdd

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ellipticAdd(N,a,b,P,Q);

Return: list L, representing the point P+Q

Note: P=(P[1]:P[2]:P[3]), Q=(Q[1]:Q[2]:Q[3]) points on the elliptic curve defined by y^2z=x^3+a*xz^2+b*z^3 over Z/N

Example:

```
LIB "crypto.lib";
bigint N=11;
bigint a=1;
bigint b=6;
list P,Q;
P[1]=2;
P[2]=4;
P[3]=1;
Q[1]=3;
Q[2]=5;
Q[3]=1;
ellipticAdd(N,a,b,P,Q);
↦ [1]:
↦    7
↦ [2]:
↦    2
↦ [3]:
↦    1
```

D.12.2.26 ellipticMult

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ellipticMult(N,a,b,P,k);

Return: a list L representing the point k*P

Note: P=(P[1]:P[2]:P[3]) a point on the elliptic curve defined by y^2z=x^3+a*xz^2+b*z^3 over Z/N

Example:

```
LIB "crypto.lib";
bigint N=11;
bigint a=1;
bigint b=6;
list P;
P[1]=2;
P[2]=4;
P[3]=1;
ellipticMult(N,a,b,P,3);
↦ [1]:
↦    8
↦ [2]:
↦    8
↦ [3]:
↦    1
```

D.12.2.27 ellipticRandomCurve

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ellipticRandomCurve(N);

Return: a list of two random numbers a,b and 4a^3+27b^2 mod N

Note: y^2z=x^3+a*xz^2+b^2*z^3 defines an elliptic curve over Z/N

Example:

```
LIB "crypto.lib";
ellipticRandomCurve(32003);
↦ [1]:
↦    22857
↦ [2]:
↦    24963
↦ [3]:
↦    1
```

D.12.2.28 ellipticRandomPoint

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ellipticRandomPoint(N,a,b);

Return: a list representing a random point (x:y:z) of the elliptic curve defined by y^2z=x^3+a*xz^2+b*z^3 over Z/N

Example:

```
LIB "crypto.lib";
ellipticRandomPoint(32003,3,181);
↦ [1]:
↦    22857
↦ [2]:
↦    17476
↦ [3]:
↦    1
```

D.12.2.29 countPoints

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: countPoints(N,a,b);

Return: the number of points of the elliptic curve defined by y^2=x^3+a*x+b over Z/N

Note: trivial approach

Example:

```
LIB "crypto.lib";
countPoints(181,71,150);
↦ 198
```

D.12.2.30 ellipticAllPoints

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ellipticAllPoints(N,a,b);

Return: list of points (x:y:z) of the elliptic curve defined by y^2z=x^3+a*xz^2+b*z^3 over Z/N

Example:

```
LIB "crypto.lib";
list L=ellipticAllPoints(181,71,150);
size(L);
↦ 198
L[size(L)];
↦ [1]:
↦    179
↦ [2]:
↦    0
↦ [3]:
↦    1
```

D.12.2.31 ShanksMestre

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ShanksMestre(q,a,b); optional: ShanksMestre(q,a,b,s); s the number of loops in the algorithm (default s=1)

Return: the number of points of the elliptic curve defined by y^2=x^3+a*x+b over Z/N

Note: algorithm of Shanks and Mestre (baby-step-giant-step)

Example:

```
LIB "crypto.lib";
ShanksMestre(32003,71,602);
↦ 32021
```

D.12.2.32 Schoof

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: Schoof(N,a,b);

Return: the number of points of the elliptic curve defined by y^2=x^3+a*x+b over Z/N

Note: algorithm of Schoof

Example:

```
LIB "crypto.lib";
Schoof(32003,71,602);
↦ 32021
```

D.12.2.33 generateG

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: generateG(a,b,m);

Return: m-th division polynomial

Note: generate the so-called division polynomials, i.e., the recursively defined polynomials p_m=generateG(a,b,m) in Z[x, y] such that, for a point (x:y:1) on the elliptic curve defined by y^2=x^3+a*x+b over Z/N the point m*P=(x-(p_(m-1)*p_(m+1))/p_m^2 :(p_(m+2)*p_(m-1)^2-p_(m-2)*p_(m+1)^2)/4y*p_m^3 :1) and m*P=0 if and only if p_m(P)=0

Example:

```
LIB "crypto.lib";
ring R = 0,(x,y),dp;
generateG(7,15,4);
↦ 4xy6+140xy4+1200xy3-980xy2-1680xy-8572x
```

D.12.2.34 factorLenstraECM

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: factorLenstraECM(N,S,B); optional: factorLenstraECM(N,S,B,d); d+1 the number of loops in the algorithm (default d=0)

Return: a factor of N or the message no factor found

Note: - computes a factor of N using Lenstra's ECM factorization
- the idea is that the fact that N is not prime is detected using the operations on the elliptic curve
- is similarly to Pollard's p-1-factorization

Example:

```
LIB "crypto.lib";
list L=primList(1000);
factorLenstraECM(181*32003,L,10,5);
↦ 181
bigint h=10;
h=h^30+25;
factorLenstraECM(h,L,4,3);
↦ 13
```

D.12.2.35 ECPP

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: ECPP(N);

Return: message:N is not prime or {L,P,m,q} as certificate for N being prime
L a list (y^2=x^3+L[1]*x+L[2] defines an elliptic curve C)
P a list ((P[1]:P[2]:P[3]) is a point of C)
m,q integers

Assume: gcd(N,6)=1

Note: The basis of the algorithm is the following theorem:
Given C, an elliptic curve over Z/N, P a point of C(Z/N), m an integer, q a prime with the following properties:
- q|m
- q>(4-th root(N) +1)^2
- m*P=0=(0:1:0)
- (m/q)*P=(x:y:z) and z a unit in Z/N
Then N is prime.

Example:

```
LIB "crypto.lib";
bigint N=1267985441;
ECPP(N);
↦ P= [1]:
↦    780306204
↦ [2]:
↦    1106324420
↦ [3]:
↦    1
↦ [1]:
↦    [1]:
↦       67394594
↦    [2]:
↦       380636642
↦ [2]:
↦    [1]:
↦       780306204
↦    [2]:
↦       1106324420
↦    [3]:
↦       1
↦ [3]:
↦    1267993236
↦ [4]:
↦    105666103
```

D.12.2.36 calculate_ordering

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: calculate_ordering(num1, primitive, mod1)

Return: x so that primitive^x == num1 mod mod1

Example:

```
LIB "crypto.lib";
bigint mod1 = 33;
bigint primitive = 14;
bigint num1 = 5;
calculate_ordering(num1,primitive,mod1);
↦ 3
```

D.12.2.37 is_primitive_root

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: is_primitive_root(primitive, mod1)

Return: 1 if primitive is a primitive root modulo mod1, 0 otherwise

Example:

```
LIB "crypto.lib";
is_primitive_root(3,7);
↦ 1
is_primitive_root(2,7);
↦ 0
```

D.12.2.38 find_first_primitive_root

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: find_first_primitive_root(mod1)

Return: First primitive root modulo mod1, 0 if no root can be found.

Example:

```
LIB "crypto.lib";
ring r = 0,x,lp;
find_first_primitive_root(7);
↦ 3
find_first_primitive_root(557);
↦ 2
```

D.12.2.39 binary_add

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: binary_add(binary_list)

Return: binary encoded list, increased by 1

Example:

```
LIB "crypto.lib";
ring r = 0,x,lp;
list binary_list = 1,0,1,1,1;
binary_add(binary_list);
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    0
↦ [4]:
↦    0
↦ [5]:
↦    0
```

D.12.2.40 inverse_modulus

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: inverse_modulus(num, mod1)

Return: inverse element of num modulo mod1

Example:

```
LIB "crypto.lib";
ring r = 0,x,lp;
int mod1 = 13;
int num = 5;
inverse_modulus(num,mod1);
↦ 8
```

D.12.2.41 is_prime

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: is_prime(n)

Return: 1 if n is prime, 0 otherwise

Example:

```
LIB "crypto.lib";
ring r = 0,x,lp;
is_prime(10);
↦ 0
is_prime(7);
↦ 1
```

D.12.2.42 find_index

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: find_index(a, e)

Return: Returns the list index of element e in list a. Returns 0 if e is not in a

Example:

```
LIB "crypto.lib";
list a = 1,5,20,6,37;
find_index(a,20);
↦ 3
find_index(a,6);
↦ 4
find_index(a,100);
↦ 0
```

D.12.2.43 subset_sum01

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: subset_sum01(knapsack,solution)

Return: binary list of the positions of the elements included in the subset sum or 0 if no solution exists

Note: This will return the first solution of the ssk-problem, given be the smallest binary encoding. It wont return several solutions if they exist

Example:

```
LIB "crypto.lib";
list h=1,4,7,32;
subset_sum01(h,20);
↦ 0
subset_sum01(h,11);
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    0
subset_sum01(h,33);
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    0
↦ [4]:
↦    1
```

D.12.2.44 subset_sum02

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: subset_sum02(knapsack,sol)

Return: binary list of the positions of the elements included in the subset sum or 0 if no solution exists

Example:

```
LIB "crypto.lib";
list h=1,4,7,32;
subset_sum02(h,20);
↦ 0
subset_sum02(h,11);
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    1
↦ [4]:
↦    0
subset_sum02(h,33);
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    0
↦ [4]:
↦    1
```

D.12.2.45 unbounded_knapsack

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: unbounded_knapsack(knapsack,profit,capacity)

Return: list of maximum profit of each iteration. For example, output_list[2] contains the maximum profit that can be achieved if the knapsack has capacity 2.

Example:

```
LIB "crypto.lib";
list h=1,4,7,32;
list knapsack = 5,2;
list profit = 10,3;
int capacity = 5;
unbounded_knapsack(knapsack,profit,capacity);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    3
↦ [4]:
↦    3
↦ [5]:
↦    6
↦ [6]:
↦    10
```

D.12.2.46 multidimensional_knapsack

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: multidimensional_knapsack(m,capacities,profits)

Return: binary list of the positions of the elements included in the optimal selection

Example:

```
LIB "crypto.lib";
ring r = 0,x,lp;
matrix m[3][3] = 1,4,10,7,8,3,1,9,7;
list c = 12,17,10;
list p = 3,2,5;
multidimensional_knapsack(m,c,p);
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    1
```

D.12.2.47 naccache_stern_generation

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: naccache_stern_generation(key, primenum)

Return: a hard knapsack list

Example:

```
LIB "crypto.lib";
naccache_stern_generation(5,292);
↦ 0
naccache_stern_generation(5,293);
↦ [1]:
↦    85
↦ [2]:
↦    164
↦ [3]:
↦    117
↦ [4]:
↦    44
```

D.12.2.48 naccache_stern_encryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: naccache_stern_encryption(knapsack, message, primenum)

Return: an encrypted message as integer

Example:

```
LIB "crypto.lib";
//Please note that the values for primenum and hardknapsack have been obtained from t
list hardknapsack = 85,164,117,44;
int primenum = 293;
list message = 1,0,1,0;
naccache_stern_encryption(hardknapsack,message,primenum);
↦ 9945
```

D.12.2.49 naccache_stern_decryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: naccache_stern_decryption(knapsack, key, primenum, message)

Return: decrypted binary list

Example:

```
LIB "crypto.lib";
//Please note that the values have been obtained from the example of naccache_stern_g
int primenum = 293;
int message = 9945;
int key = 5;
list hardknapsack = 85,164,117,44;
naccache_stern_decryption(hardknapsack,key,primenum,message);
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    1
↦ [4]:
↦    0
```

D.12.2.50 m_merkle_hellman_transformation

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: m_merkle_hellman_transformation(knapsack, primitive, mod1)

Return: list containing a hard knapsack

Example:

```
LIB "crypto.lib";
//Please note that the values for primenum and hardknapsack have been obtained from t
list knapsack = 2,3,5,7;
int mod1 = 211;
int primitive = 2;
m_merkle_hellman_transformation(knapsack,primitive,mod1);
↦ [1]:
↦    1
↦ [2]:
↦    43
↦ [3]:
↦    132
↦ [4]:
↦    139
```

D.12.2.51 m_merkle_hellman_encryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: m_merkle_hellman_encryption(knapsack, message)

Return: an encrypted message as integer

Note: This works in the same way as merkle_hellman_encryption. The additional function is created to keep consistency with the needed functions for every kryptosystem.

Example:

```
LIB "crypto.lib";
//Please note that the values for primenum and hardknapsack have been obtained from t
list knapsack = 1,43,132,139;
list message = 1,0,0,1;
m_merkle_hellman_encryption(knapsack,message);
↦ 140
```

D.12.2.52 m_merkle_hellman_decryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: m_merkle_hellman_decryption(knapsack, primitive, mod1, message)

Return: decrypted binary list

Example:

```
LIB "crypto.lib";
//Please note that the values  have been obtained from the example of m_merkle_hellma
list knapsack = 2,3,5,7;
int message = 140;
bigint primitive = 2;
bigint mod1 = 211;
m_merkle_hellman_decryption(knapsack,primitive,mod1,message);
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    0
↦ [4]:
↦    1
```

D.12.2.53 merkle_hellman_encryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: merkle_hellman_encryption(knapsack, message)

Return: encrypted integer

Example:

```
LIB "crypto.lib";
//Please note that the values  have been obtained from the example of merkle_hellman_
list hardknapsack =3,9,15,13;
list message = 0,1,0,1;
merkle_hellman_encryption(hardknapsack,message);
↦ 22
```

D.12.2.54 merkle_hellman_decryption

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: merkle_hellman_decryption(knapsack, key, mod1, message)

Return: decrypted binary list

Example:

```
LIB "crypto.lib";
//Please note that the values  have been obtained from the example of merkle_hellman_
list hardknapsack =3,9,15,13;
int key = 3;
int message = 22;
int mod1 = 23;
merkle_hellman_decryption(hardknapsack, key, mod1, message);
↦ 15
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
```

```
↦    0
↦ [4]:
↦    1
```

D.12.2.55 super_increasing_knapsack

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: super_increasing_knapsack(ksize)

Return: super-increasing knapsack list

Example:

```
LIB "crypto.lib";
super_increasing_knapsack(10);
↦ [1]:
↦    512
↦ [2]:
↦    256
↦ [3]:
↦    128
↦ [4]:
↦    64
↦ [5]:
↦    32
↦ [6]:
↦    16
↦ [7]:
↦    8
↦ [8]:
↦    4
↦ [9]:
↦    2
↦ [10]:
↦    1
```

D.12.2.56 h_increasing_knapsack

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: h_increasing_knapsack(ksize, h)

Return: h-increasing knapsack list

Example:

```
LIB "crypto.lib";
h_increasing_knapsack(10,5);
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    4
↦ [4]:
↦    8
↦ [5]:
```

```
↦    16
↦ [6]:
↦    32
↦ [7]:
↦    63
↦ [8]:
↦    124
↦ [9]:
↦    244
↦ [10]:
↦    480
```

D.12.2.57 injective_knapsack

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: injective_knapsack(ksize, kmaxelement)

Return: list of injective knapsacks with maximal element kmaxelement and size ksize

Example:

```
LIB "crypto.lib";
injective_knapsack(3,9);
↦ [1]:
↦    [1]:
↦       9
↦    [2]:
↦       1
↦    [3]:
↦       3
↦ [2]:
↦    [1]:
↦       9
↦    [2]:
↦       1
↦    [3]:
↦       4
↦ [3]:
↦    [1]:
↦       9
↦    [2]:
↦       1
↦    [3]:
↦       5
↦ [4]:
↦    [1]:
↦       9
↦    [2]:
↦       1
↦    [3]:
↦       6
↦ [5]:
↦    [1]:
↦       9
↦    [2]:
↦       1
↦    [3]:
↦       7
↦ [6]:
↦    [1]:
↦       9
↦    [2]:
↦       2
↦    [3]:
↦       1
↦ [7]:
↦    [1]:
↦       9
↦    [2]:
↦       2
↦    [3]:
↦       3
↦ [8]:
↦    [1]:
↦       9
↦    [2]:
↦       2
↦    [3]:
↦       4
↦ [9]:
↦    [1]:
↦       9
↦    [2]:
↦       2
↦    [3]:
↦       5
↦ [10]:
↦    [1]:
↦       9
↦    [2]:
↦       2
↦    [3]:
↦       6
↦ [11]:
↦    [1]:
↦       9
↦    [2]:
↦       3
↦    [3]:
↦       4
↦ [12]:
↦    [1]:
↦       9
↦    [2]:
↦       3
↦    [3]:
↦       5
↦ [13]:
↦    [1]:
↦       9
↦    [2]:
↦       3
↦    [3]:
↦       7
↦ [14]:
↦    [1]:
↦       9
↦    [2]:
↦       4
↦    [3]:
↦       6
↦ [15]:
↦    [1]:
↦       9
↦    [2]:
↦       4
↦    [3]:
↦       7
↦ [16]:
↦    [1]:
↦       9
↦    [2]:
↦       5
↦    [3]:
↦       6
↦ [17]:
↦    [1]:
↦       9
↦    [2]:
↦       5
↦    [3]:
↦       7
↦ [18]:
↦    [1]:
↦       9
↦    [2]:
↦       6
↦    [3]:
↦       7
↦ [19]:
↦    [1]:
↦       9
↦    [2]:
↦       8
↦    [3]:
↦       2
↦ [20]:
↦    [1]:
↦       9
↦    [2]:
```

```
↦       8
↦    [3]:
↦       3
↦ [21]:
↦    [1]:
↦       9
↦    [2]:
↦       8
↦    [3]:
↦       4
↦ [22]:
↦    [1]:
↦       9
↦    [2]:
↦       8
↦    [3]:
↦       5
↦ [23]:
↦    [1]:
↦       9
↦    [2]:
↦       8
↦    [3]:
↦       6
↦ [24]:
↦    [1]:
↦       9
↦    [2]:
↦       8
↦    [3]:
↦       7
```

D.12.2.58 calculate_max_sum

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: calculate_max_sum(a)

Return: sum of all elements in a

Example:

```
LIB "crypto.lib";
list a = 1,5,3,2,12;
calculate_max_sum(a);
↦ 23
```

D.12.2.59 set_is_injective

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: set_is_injective(a)

Return: 1 if a is injective, 0 otherwise

Example:

```
LIB "crypto.lib";
list inj = 1,5,7,41;
list non_inj = 1,2,3,4;
set_is_injective(inj);
↦ 1
set_is_injective(non_inj);
↦ 0
```

D.12.2.60 is_h_injective

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: is_h_injective(a, h)

Return: 1 if a is h-injective, 0 otherwise

Example:

```
LIB "crypto.lib";
list h_inj = 1,2,4,10,17;
is_h_injective(h_inj,3);
↦ 1
//1+2+4+10=17
is_h_injective(h_inj,4);
↦ 0
```

D.12.2.61 is_fix_injective

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: is_fix_injective(a)

Return: 1 if a is fix-injective, 0 otherwise

Example:

```
LIB "crypto.lib";
//this is fix-injective because 17=10+2+4+1 with different numbers of addens.
list fix_inj = 1,2,4,10,17;
//this is not fix-injective because 4+1=2+3.
list not_fix_inj = 1,2,3,4;
is_fix_injective(fix_inj);
↦ 1
is_fix_injective(not_fix_inj);
↦ 0
```

D.12.2.62 three_elements

Procedure from library `crypto.lib` (see Section D.12.2 [crypto_lib], page 1988).

Usage: three_elements(out, iterations)

Return: Injective_knapsack created with the three elements method

Example:

```
LIB "crypto.lib";
//this is fix-injective because 17=10+2+4+1 with different numbers of addens.
list super_increasing = 1,2,4,10,20;
list a = three_elements(super_increasing,2);
```

```
a;
↦ [1]:
↦    1529
↦ [2]:
↦    1147
↦ [3]:
↦    764
↦ [4]:
↦    153
↦ [5]:
↦    115
↦ [6]:
↦    76
↦ [7]:
↦    20
↦ [8]:
↦    10
↦ [9]:
↦    4
↦ [10]:
↦    2
↦ [11]:
↦    1
set_is_injective(a);
↦ 1
```

D.12.3 hyperel_lib

Library: hyperel.lib

Author: Markus Hochstetter, markushochstetter@gmx.de

Note: The library provides procedures for computing with divisors in the jacobian of hyperelliptic curves. In addition procedures are available for computing the rational representation of divisors and vice versa. The library is intended to be used for teaching and demonstrating purposes but not for efficient computations.

Procedures:

D.12.3.1 ishyper

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: ishyper(h,f); h,f=poly

Return: 1 if y^2+h(x)y=f(x) is hyperelliptic, 0 otherwise

Note: Tests, if y^2+h(x)y=f(x) is a hyperelliptic curve.
Curve is defined over basering. Additionally shows error-messages.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
```

```
ishyper(h,f);
↦ 1
```

D.12.3.2 isoncurve

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: isoncurve(P,h,f); h,f=poly; P=list

Return: 1 or 0 (if P is on curve or not)

Note: Tests, if P=(P[1],P[2]) is on the hyperelliptic curve y^2+h(x)y=f(x). Curve is defined over basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
list P=2,3;
isoncurve(P,h,f);
↦ 1
```

D.12.3.3 chinrestp

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: chinrestp(b,moduli); moduli, b, moduli=list of polynomials

Return: poly x, s.t. x= b[i] mod moduli[i]

Note: chinese remainder theorem for polynomials

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
list b=3x-4, -3x2+1, 1, 4;
list moduli=(x-2)^2, (x-5)^3, x-1, x-6;
chinrestp(b,moduli);
↦ -x6-3x5-x4+2x3-2x2+3x+3
```

D.12.3.4 norm

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: norm(a,b,h,f);

Return: norm of a(x)-b(x)y in IF[C]

Note: The norm is a polynomial in just one variable.
Curve C: y^2+h(x)y=f(x) is defined over basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
poly a=x2+1;
poly b=x;
norm(a,b,h,f);
↦ -x7+2x6+3x4-x3+1
```

D.12.3.5 multi

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: multi(a,b,c,d,h,f);

Return: list L with L[1]-L[2]y=(a(x)-b(x)y)*(c(x)-d(x)y) in IF[C]

Note: Curve C: y^2+h(x)y=f(x) is defined over basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
poly h=x;
poly f=x5+5x4+6x2+x+3;
// hyperelliptic curve y^2 + h*y = f
poly a=x2+1;
poly b=x;
poly c=5;
poly d=-x;
multi(a,b,c,d,h,f);
↦ [1]:
↦    -x7+2x6+x4-x3+2x2-2
↦ [2]:
↦    -2x3-3x
```

D.12.3.6 divisor

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: divisor(a,b,h,f); optional: divisor(a,b,h,f,s); s=0,1

Return: list P

Note: P[1][3]*(P[1][1], P[1][2]) +...+ P[size(P)][3]*
*(P[size(P)][1], P[size(P)][2]) - (*)infty=div(a(x)-b(x)y) if there is an optional parameter s!=0, then divisor additionally returns a parameter, which says, whether irreducible polynomials occurred during computations or not. Otherwise only warnings are displayed on the monitor. For s=0 nothing happens.
Curve C: y^2+h(x)y=f(x) is defined over basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
poly a=(x-1)^2*(x-6);
poly b=0;
divisor(a,b,h,f,1);
```

```
↦ [1]:
↦    [1]:
↦       -1
↦    [2]:
↦       -3
↦    [3]:
↦       2
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       -2
↦    [3]:
↦       2
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       2
↦ 0
```

D.12.3.7 gcddivisor

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: gcddivisor(p,q);

Return: list P

Note: gcd of two divisors

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// two divisors
list p=list(-1,-3,1),list(1,1,2);
list q=list(1,1,1),list(2,2,1);
gcddivisor(p,q);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.12.3.8 semidiv

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: semidiv(D,h,f);

Return: list P

Note: important: Divisor D has to be semireduced!
Computes semireduced divisor P[1][3]*(P[1][1], P[1][2]) +...+ P[size(P)][3]* *(P[size(P)][1], P[size(P)][2]) - (*)infty=div(D[1],D[2])
Curve C:y^2+h(x)y=f(x) is defined over basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// Divisor
list D=x2-1,2x-1;
semidiv(D,h,f);
↦ [1]:
↦    [1]:
↦       -1
↦    [2]:
↦       -3
↦    [3]:
↦       1
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.12.3.9 cantoradd

Procedure from library hyperel.lib (see Section D.12.3 [hyperel_lib], page 2015).

Usage: cantoradd(D,Q,h,f);

Return: list P

Note: Cantor's Algorithm - composition
important: D and Q have to be semireduced!
Computes semireduced divisor div(P[1],P[2])= div(D[1],D[2]) + div(Q[1],Q[2]) The divisors are defined over the basering.
Curve C: y^2+h(x)y=f(x) is defined over the basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// two divisors in rational representation
list D=x2-1,2x-1;
list Q=x2-3x+2,-3x+1;
cantoradd(D,Q,h,f);
```

```
↦ [1]:
↦    x2-x-2
↦ [2]:
↦    -3x+1
```

D.12.3.10 cantorred

Procedure from library hyperel.lib (see Section D.12.3 [hyperel_lib], page 2015).

Usage: cantorred(D,h,f);

Return: list N

Note: Cantor's algorithm - reduction.
important: Divisor D has to be semireduced!
Computes reduced divisor div(N[1],N[2])= div(D[1],D[2]).
The divisors are defined over the basering.
Curve C: y^2+h(x)y=f(x) is defined over the basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// semireduced divisor
list D=2x4+3x3-3x-2, -x3-2x2+3x+1;
cantorred(D,h,f);
↦ [1]:
↦    x2-2x+2
↦ [2]:
↦    2x-2
```

D.12.3.11 double

Procedure from library hyperel.lib (see Section D.12.3 [hyperel_lib], page 2015).

Usage: double(D,h,f);

Return: list Q=2*D

Note: important: Divisor D has to be semireduced!
Special case of Cantor's algorithm.
Computes reduced divisor div(Q[1],Q[2])= 2*div(D[1],D[2]).
The divisors are defined over the basering.
Curve C:y^2+h(x)y=f(x) is defined over the basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// reduced divisor
list D=x2-1,2x-1;
double(D,h,f);
```

```
↦ [1]:
↦    x2-2x+1
↦ [2]:
↦    3x-2
```

D.12.3.12 cantormult

Procedure from library `hyperel.lib` (see Section D.12.3 [hyperel_lib], page 2015).

Usage: cantormult(m,D,h,f);

Return: list res=m*D

Note: important: Divisor D has to be semireduced!
Uses repeated doublings for a faster computation
of the reduced divisor m*D.
Attention: Factor m=int, this means bounded.
For m<0 the inverse of m*D is returned.
The divisors are defined over the basering.
Curve C: y^2+h(x)y=f(x) is defined over the basering.

Example:

```
LIB "hyperel.lib";
ring R=7,x,dp;
// hyperelliptic curve y^2 + h*y = f
poly h=x;
poly f=x5+5x4+6x2+x+3;
// reduced divisor
list D=x2-1,2x-1;
cantormult(34,D,h,f);
↦ [1]:
↦    x2-3x-3
↦ [2]:
↦    x+1
```

D.12.4 teachstd_lib

Library: teachstd.lib

Purpose: Procedures for teaching standard bases

Author: G.-M. Greuel, greuel@mathematik.uni-kl.de

Note: The library is intended to be used for teaching purposes, but not for serious computations. Sufficiently high printlevel allows to control each step, thus illustrating the algorithms at work. The procedures are implemented exactly as described in the book 'A SINGULAR Introduction to Commutative Algebra' by G.-M. Greuel and G. Pfister (Springer 2002).

Procedures:

D.12.4.1 ecart

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: ecart(f); f poly or vector

Return: the ecart e of f of type int

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),ls;
ecart((y+z+x+xyz)**2);
↦ 4
ring s=0,(x,y,z),dp;
ecart((y+z+x+xyz)**2);
↦ 0
```

D.12.4.2 tail

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: tail(f); f poly or vector

Return: f-lead(f), the tail of f of type poly

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),ls;
tail((y+z+x+xyz)**2);
↦ 2yz+y2+2xz+2xy+2xyz2+2xy2z+x2+2x2yz+x2y2z2
ring s=0,(x,y,z),dp;
tail((y+z+x+xyz)**2);
↦ 2x2yz+2xy2z+2xyz2+x2+2xy+y2+2xz+2yz+z2
```

D.12.4.3 sameComponent

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: sameComponent(f,g); f,g poly or vector

Return: 1 if f and g are of type poly or if f and g are of type vector and their leading monomials involve the same module component, 0 if not

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
sameComponent([y+z+x,xyz],[z2,xyz]);
↦ 1
sameComponent([y+z+x,xyz],[z4,xyz]);
↦ 0
sameComponent(y+z+x+xyz, xy+z5);
↦ 1
```

D.12.4.4 leadmonomial

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: leadmonomial(f); f poly or vector

Return: the leading monomial of f of type poly

Note: if f is of type poly, leadmonomial(f)=leadmonom(f), if f is of type vector and if leadmonom(f)=m*gen(i) then leadmonomial(f)=m

Example:

```
LIB "teachstd.lib";
ring s=0,(x,y,z),(c,dp);
leadmonomial((y+z+x+xyz)^2);
↦ x2y2z2
leadmonomial([(y+z+x+xyz)^2,xyz5]);
↦ x2y2z2
```

D.12.4.5 monomialLcm

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: monomialLcm(m,n); m,n of type poly or vector

Return: least common multiple of leading monomials of m and n, of type poly

Note: if m = (x1...xr)^(a1,...,ar)*gen(i) (gen(i)=1 if m is of type poly) and n = (x1...xr)^(b1,...,br)*gen(j), then the proc returns (x1,...,xr)^(max(a1,b1),...,max(ar,br)) if i=j and 0 if i!=j.

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),ds;
monomialLcm(xy2,yz3);
↦ xy2z3
monomialLcm([xy2,xz],[yz3]);
↦ 0
monomialLcm([xy2,xz3],[yz3]);
↦ xy2z3
```

D.12.4.6 spoly

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: spoly(f,g[,s]); f,g poly or vector, s int

Return: the s-polynomial of f and g, of type poly or vector
if s!=0 the symmetric s-polynomial (without division) is returned

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),ls;
spoly(2x2+x2y,3y3+xyz);
↦ x2y4-2/3x3yz
ring s=0,(x,y,z),(c,dp);
spoly(2x2+x2y,3y3+xyz);
↦ -1/3x3yz+2x2y2
spoly(2x2+x2y,3y3+xyz,1);           //symmetric s-poly without division
↦ -x3yz+6x2y2
spoly([5x2+x2y,z5],[x2,y3,y4]);     //s-poly for vectors
↦ [5x2,z5-y4,-y5]
```

D.12.4.7 minEcart

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: minEcart(T,h); T ideal or module, h poly or vector

Return: element g from T such that leadmonom(g) divides leadmonom(h)
ecart(g) is minimal with this property (if T != 0);
return 0 if T is 0 or h = 0

Example:

```
LIB "teachstd.lib";
ring R=0,(x,y,z),dp;
ideal T = x2y+x2,y3+xyz,xyz2+z4;
poly h = x2y2z2+x5+yx3+z6;
minEcart(T,h);"";
↦ x2y+x2
↦
ring S=0,(x,y,z),(c,ds);
module T = [x2+x2y,y2],[y3+xyz,x3-z3],[x3y+z4,0,x2];
vector h = [x3y+x5+x2y2z2+z6,x3];
minEcart(T,h);
↦ [x3y+z4,0,x2]
```

D.12.4.8 NFMora

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: NFMora(f,G[,s]); f poly or vector,G ideal or module, s int

Return: the Mora normal form of f w.r.t. G, same type as f
if s!=0 the symmetric s-polynomial (without division) is used

Note: Show comments if printlevel > 0, pauses computation if printlevel > 1

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
poly f = x2y2z2+x5+yx3+z6-3y3;
ideal G = x2y+x2,y3+xyz,xyz2+z6;
NFMora(f,G);"";
↦ x5-x2yz2+x3y-xyz2-3y3
↦
ring s=0,(x,y,z),ds;
poly f = x3y+x5+x2y2z2+z6;
ideal G = x2+x2y,y3+xyz,x3y2+z4;
NFMora(f,G);"";
↦ 0
↦
vector v = [f,x2+x2y];
module M = [x2+x2y,f],[y3+xyz,y3],[x3y2+z4,z2];
NFMora(v,M);
↦ x2*gen(2)+x2y*gen(2)+x3y*gen(1)+x5*gen(1)+x2y2z2*gen(1)+z6*gen(1)
```

D.12.4.9 prodcrit

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: prodcrit(f,g[,o]); f,g poly or vector, and optional int argument o

Return: 1 if product criterion applies in the same module component, 2 if lead(f) and lead(g) involve different components, 0 else

Note: if product criterion applies we can delete (f,g) from pairset. This procedure returns 0 if o is given and is a positive integer, or you may set the attribute "default_arg" for prodcrit to 1.

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
poly f = y3z3+x5+yx3+z6;
poly g = x5+yx3;
prodcrit(f,g);
↦ 1
vector v = x3z2*gen(1)+x3y*gen(1)+x2y*gen(2);
vector w = y4*gen(1)+y3*gen(2)+xyz*gen(1);
prodcrit(v,w);
↦ 0
```

D.12.4.10 chaincrit

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: chaincrit(f,g,h); f,g,h poly or module

Return: 1 if chain criterion applies, 0 else

Note: if chain criterion applies to f,g,h we can delete (g,h) from pairset

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
poly f = x2y2z2+x5+yx3+z6;
poly g = x5+yx3;
poly h = y2z5+x5+yx3;
chaincrit(f,g,h);
↦ 1
vector u = [x2y3-z2,x2y];
vector v = [x2y2+z2,x2-y2];
vector w = [x2y4+z3,x2+y2];
chaincrit(u,v,w);
↦ 1
```

D.12.4.11 pairset

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: pairset(G); G ideal or module

Return: list L,
L[1] = the pairset of G as list (not containing pairs for which the product or the chain criterion applies),
L[2] = intvec v, v[1]= # product criterion, v[2]= # chain criterion

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
ideal G = x2y+x2,y3+xyz,xyz2+z4;
pairset(G);"";
↦ [1]:
↦    [1]:
↦       _[1]=x2y+x2
↦       _[2]=y3+xyz
↦    [2]:
↦       _[1]=x2y+x2
↦       _[2]=xyz2+z4
↦    [3]:
↦       _[1]=y3+xyz
↦       _[2]=xyz2+z4
↦ [2]:
↦    0,0
↦
module T = [x2y3-z2,x2y],[x2y2+z2,x2-y2],[x2y4+z3,x2+y2];
pairset(T);
↦ [1]:
↦    [1]:
↦       _[1]=x2y3*gen(1)+x2y*gen(2)-z2*gen(1)
↦       _[2]=x2y2*gen(1)+x2*gen(2)-y2*gen(2)+z2*gen(1)
↦    [2]:
↦       _[1]=x2y3*gen(1)+x2y*gen(2)-z2*gen(1)
↦       _[2]=x2y4*gen(1)+z3*gen(1)+x2*gen(2)+y2*gen(2)
↦ [2]:
↦    0,1
```

D.12.4.12 updatePairs

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: updatePairs(P,S,h); P list, S ideal or module, h poly or vector
P a list of pairs of polys or vectors (obtained from pairset)

Return: list Q,
Q[1] = the pairset P enlarged by all pairs (f,h), f from S, without pairs for which the product or the chain criterion applies
Q[2] = intvec v, v[1]= # product criterion, v[2]= # chain criterion

Example:

```
LIB "teachstd.lib";
ring R1=0,(x,y,z),(c,dp);
ideal S = x2y+x2,y3+xyz;
poly h = x2y+xyz;
list P = pairset(S)[1];
P;"";
↦ [1]:
↦    _[1]=x2y+x2
↦    _[2]=y3+xyz
↦
updatePairs(P,S,h);"";
```

```
↦ [1]:
↦    [1]:
↦       _[1]=x2y+x2
↦       _[2]=y3+xyz
↦    [2]:
↦       _[1]=x2y+x2
↦       _[2]=x2y+xyz
↦ [2]:
↦    0,1
↦ 
module T = [x2y3-z2,x2y],[x2y4+z3,x2+y2];
P = pairset(T)[1];
P;"";
↦ [1]:
↦    _[1]=[x2y3-z2,x2y]
↦    _[2]=[x2y4+z3,x2+y2]
↦ 
updatePairs(P,T,[x2+x2y,y3+xyz]);
↦ [1]:
↦    [1]:
↦       _[1]=[x2y3-z2,x2y]
↦       _[2]=[x2y4+z3,x2+y2]
↦    [2]:
↦       _[1]=[x2y3-z2,x2y]
↦       _[2]=[x2y+x2,y3+xyz]
↦ [2]:
↦    0,1
```

D.12.4.13 standard

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: standard(i[,s]); id ideal or module, s int

Return: a standard basis of id, using generalized Mora's algorithm which is Buchberger's algorithm for global monomial orderings. If s!=0 the symmetric s-polynomial (without division) is used

Note: Show comments if printlevel > 0, pauses computation if printlevel > 1

Example:

```
LIB "teachstd.lib";
ring r=0,(x,y,z),dp;
ideal G = x2y+x2,y3+xyz,xyz2+z4;
standard(G);"";
↦ _[1]=x2y+x2
↦ _[2]=y3+xyz
↦ _[3]=xyz2+z4
↦ _[4]=x3z-x2y
↦ _[5]=-xz4+x2z2
↦ _[6]=-y2z4-x2z3
↦ _[7]=z6+x2yz2
↦ _[8]=x2z3-x3z
↦ _[9]=-x2z2+x3
↦ _[10]=x4-x2yz
```

```
↦
ring s=0,(x,y,z),(c,ds);
ideal G = 2x2+x2y,y3+xyz,3x3y+z4;
standard(G);"";
↦ _[1]=2x2+x2y
↦ _[2]=y3+xyz
↦ _[3]=3x3y+z4
↦ _[4]=-2/3z4+x3y2
↦
standard(G,1);"";           //use symmetric s-poly without division
↦ _[1]=2x2+x2y
↦ _[2]=y3+xyz
↦ _[3]=3x3y+z4
↦ _[4]=-2z4+3x3y2
↦
module M = [2x2,x3y+z4],[3y3+xyz,y3],[5z4,z2];
standard(M);
↦ _[1]=[2x2,x3y+z4]
↦ _[2]=[3y3+xyz,y3]
↦ _[3]=[5z4,z2]
↦ _[4]=[0,-2/3x2y3+x3y4+1/3x4y2z+y3z4+1/3xyz5]
↦ _[5]=[0,-2/5x2z2+x3yz4+z8]
↦ _[6]=[0,-3/5y3z2-1/5xyz3+y3z4]
```

D.12.4.14 localstd

Procedure from library `teachstd.lib` (see Section D.12.4 [teachstd_lib], page 2021).

Usage: localstd(id); id = ideal

Return: A standard basis for a local degree ordering, using Lazard's method.

Note: The procedure homogenizes id w.r.t. a new 1st variable local@t, computes a SB w.r.t. (dp(1),dp) and substitutes local@t by 1. Hence the result is a SB with respect to an ordering which sorts first w.r.t. the subdegree of the original variables and then refines it with dp. This is the local degree ordering ds.
localstd may be used in order to avoid cancellation of units and thus to be able to use option(contentSB) also for local orderings.

Example:

```
LIB "teachstd.lib";
ring R = 0,(x,y,z),ds;
ideal i  = xyz+z5,2x2+y3+z7,3z5+y5;
localstd(i);
↦ _[1]=y5+3z5
↦ _[2]=3x4y3z8-4x3y3z9+6x2y4z9+3y5z10
↦ _[3]=3x4z13-4x3z14+6x2yz14+3y2z15
↦ _[4]=3x4yz12-4x3yz13+6x2y2z13+3y3z14
↦ _[5]=2x2z9+x2y2z8+y3z9
↦ _[6]=2x2y4z5+y7z5-3x2yz9
↦ _[7]=6y2z10-3x2y3z8+4xy3z9-3y4z9
↦ _[8]=3x2y2z8+3y3z9+2xy4z8
↦ _[9]=18z14-4xy6z8+3y7z8-9x2yz12
↦ _[10]=xyz+z5
↦ _[11]=3xz6-y4z5
↦ _[12]=3y3z6+2xy4z5-3xyz9
↦ _[13]=y4z5-2xz9-xy2z8
↦ _[14]=3z10+2xyz9+xy3z8
↦ _[15]=2x2z5+y3z5-xyz8
↦ _[16]=y4z-2xz5+yz8
↦ _[17]=3z6+2xyz5-y2z8
↦ _[18]=2x2+y3+z7
```

D.12.5 weierstr_lib

Library: weierstr.lib

Purpose: Procedures for the Weierstrass Theorems

Author: G.-M. Greuel, greuel@mathematik.uni-kl.de

Procedures:

D.12.5.1 weierstrDiv

Procedure from library `weierstr.lib` (see Section D.12.5 [weierstr_lib], page 2029).

Usage: weierstrDiv(g,f,d); g,f=poly, d=integer

Assume: f must be general of finite order, say b, in the last ring variable, say T; if not use the procedure lastvarGeneral first

Purpose: perform the Weierstrass division of g by f up to order d

Return: - a list, say l, of two polynomials and an integer, such that
g = l[1]*f + l[2], deg_T(l[2]) < b, up to (including) total degree d
- l[3] is the number of iterations used
- if f is not T-general, return (0,g)

Note: the procedure works for any monomial ordering

Theory: the proof of Grauert-Remmert (Analytische Stellenalgebren) is used for the algorithm

Example:

```
LIB "weierstr.lib";
ring R = 0,(x,y),ds;
poly f = y - xy2 + x2;
poly g = y;
list l = weierstrDiv(g,f,10); l;"";
↦ [1]:
↦    1+xy-x3+x2y2-2x4y+2x6+x3y3-3x5y2+5x7y+x4y4-5x9-4x6y3+9x8y2+x5y5
↦ [2]:
↦    -x2+x5-2x8
↦ [3]:
↦    5
↦
l[1]*f + l[2];                //g = l[1]*f+l[2] up to degree 10
↦ y-5x11+14x10y2+5x7y5-9x9y4-x6y7
```

D.12.5.2 weierstrPrep

Procedure from library `weierstr.lib` (see Section D.12.5 [weierstr_lib], page 2029).

Usage: weierstrPrep(f,d); f=poly, d=integer

Assume: f must be general of finite order, say b, in the last ring variable, say T; if not apply the procedure lastvarGeneral first

Purpose: perform the Weierstrass preparation of f up to order d

Return: - a list, say l, of two polynomials and one integer,
l[1] a unit, l[2] a Weierstrass polynomial, l[3] an integer such that l[1]*f = l[2], where l[2] is a Weierstrass polynomial, (i.e. l[2] = T^b + lower terms in T) up to (including) total degree d l[3] is the number of iterations used
- if f is not T-general, return (0,0)

Note: the procedure works for any monomial ordering

Theory: the proof of Grauert-Remmert (Analytische Stellenalgebren) is used for the algorithm

Example:

```
LIB "weierstr.lib";
ring R = 0,(x,y),ds;
poly f = xy+y2+y4;
list l = weierstrPrep(f,5); l; "";
↦ [1]:
↦    1-x2+xy-y2+3x4-3x3y+3x2y2-2xy3+y4
↦ [2]:
↦    xy+y2-x3y
↦ [3]:
↦    6
↦
f*l[1]-l[2];                        // = 0 up to degree 5
↦ 3x5y+3x4y4-3x3y5+3x2y6-2xy7+y8
```

D.12.5.3 lastvarGeneral

Procedure from library `weierstr.lib` (see Section D.12.5 [weierstr_lib], page 2029).

Usage: lastvarGeneral(f,d); f=poly

Return: poly, say g, obtained from f by a generic change of variables, s.t. g is general of finite order b w.r.t. the last ring variable, say T (i.e. g(0,...,0,T)= c*T^b + higher terms, c!=0)

Note: the procedure works for any monomial ordering

Example:

```
LIB "weierstr.lib";
ring R = 2,(x,y,z),ls;
poly f = xyz;
lastvarGeneral(f);
↦ z24+yz19+xz6+xyz
```

D.12.5.4 generalOrder

Procedure from library `weierstr.lib` (see Section D.12.5 [weierstr_lib], page 2029).

Usage: generalOrder(f); f=poly

Return: integer b if f is general of order b w.r.t. the last variable, say T, resp. -1 if not (i.e. f(0,...,0,T) is of order b, resp. f(0,...,0,T)==0)

Note: the procedure works for any monomial ordering

Example:

```
LIB "weierstr.lib";
ring R = 0,(x,y),ds;
poly f = x2-4xy+4y2-2xy2+4y3+y4;
generalOrder(f);
↦ 2
```

D.12.6 rootsmr_lib

Library: rootsmr.lib

Purpose: Counting the number of real roots of polynomial systems

Author: Enrique A. Tobis, etobis@dc.uba.ar

Overview: Routines for counting the number of real roots of a multivariate polynomial system. Two methods are implemented: deterministic computation of the number of roots, via the signature of a certain bilinear form (nrRootsDeterm); and a rational univariate projection, using a pseudorandom polynomial (nrRootsProbab). It also includes a command to verify the correctness of the pseudorandom answer.

References:

Basu, Pollack, Roy, "Algorithms in Real Algebraic Geometry", Springer, 2003.

Procedures:

D.12.6.1 nrRootsProbab

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Return: int: the number of real roots of the ideal I by a probabilistic algorithm

Assume: If I is not a Groebner basis, then a Groebner basis will be computed by using std. If I is already a Groebner basis (i.e. if attrib(I,"isSB"); returns 1) then this Groebner basis will be used, hence it must be one w.r.t. (any) global ordering. This may be useful if the ideal is known to be a Groebner basis or if it can be computed faster by a different method.

Note: If n<10 is given, n is the number of digits being used for constructing a random characteristic polynomial, a bigger n is more safe but slower (default: n=5).
If printlevel>0 the number of complex solutions is displayed (default: printlevel=0).

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z),lp;
ideal i = (x-1)*(x-2),(y-1)^3*(x-y),(z-1)*(z-2)*(z-3)^2;
nrRootsProbab(i);       //no of real roots (using internally std)
```

```
↦ 9
i = groebner(i);        //using the hilbert driven GB computation
int pr = printlevel;
printlevel = 2;
nrRootsProbab(i);
↦ //ideal has 32 complex solutions, counted with multiplicity
↦ *********************************************************************
↦ * WARNING: This polynomial was obtained using  pseudorandom numbers.*
↦ * If you want to verify the result, please use the command          *
↦ *                                                                   *
↦ * verify(p,b,i)                                                     *
↦ *                                                                   *
↦ * where p is the polynomial I returned, b is the monomial basis     *
↦ * used, and i the Groebner basis of the ideal                       *
↦ *********************************************************************
↦ 9
printlevel = pr;
```

See also: Section D.12.6.2 [nrRootsDeterm], page 2032; Section D.12.7.15 [nrroots], page 2046; Section D.12.6.9 [randcharpoly], page 2036; Section D.8.4.2 [solve], page 1844.

D.12.6.2 nrRootsDeterm

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Return: int: the number of real roots of the ideal I by a deterministic algorithm

Assume: If I is not a Groebner basis, then a Groebner basis will be computed by using std. If I is already a Groebner basis (i.e. if attrib(I,"isSB"); returns 1) then this Groebner basis will be used, hence it must be one w.r.t. (any) global ordering. This may be useful if the ideal is known to be a Groebner basis or if it can be computed faster by a different method.

Note: If printlevel>0 the number of complex solutions is displayed (default: printlevel=0). The procedure nrRootsProbab is usually faster.

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z),lp;
ideal I = (x-1)*(x-2),(y-1),(z-1)*(z-2)*(z-3)^2;
nrRootsDeterm(I);       //no of real roots (using internally std)
↦ 6
I = groebner(I);        //using the hilbert driven GB computation
int pr = printlevel;
printlevel = 2;
nrRootsDeterm(I);
↦ //ideal has 8 complex solutions, counted with multiplicity
↦ 6
printlevel = pr;
```

See also: Section D.12.6.1 [nrRootsProbab], page 2031; Section D.12.7.15 [nrroots], page 2046; Section D.8.4.2 [solve], page 1844; Section D.12.6.4 [sturmquery], page 2033.

D.12.6.3 symsignature

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: symsignature(m); m matrix. m must be symmetric.

Return: int: the signature of m

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
matrix m = matbil(1,b,i);
symsignature(m);
↦ 4
```

See also: Section D.12.6.5 [matbil], page 2033; Section D.12.6.4 [sturmquery], page 2033.

D.12.6.4 sturmquery

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: sturmquery(h,b,i); h poly, b,i ideal

Return: int: the Sturm query of h in V(i)

Assume: i is a Groebner basis, b is an ordered monomial basis of r/i, r = basering.

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
sturmquery(1,b,i);
↦ 4
```

See also: Section D.12.6.5 [matbil], page 2033; Section D.12.6.3 [symsignature], page 2032.

D.12.6.5 matbil

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: matbil(h,b,i); h poly, b,i ideal

Return: matrix: the matrix of the bilinear form (f,g) `|->` trace(m_fhg), m_fhg = multiplication with fhg on r/i

Assume: i is a Groebner basis and b is an ordered monomial basis of r/i, r = basering

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
poly f = x3-xy+y-13+x4-y2x;
matrix m = matbil(f,b,i);
print(m);
↦ 0,     13182, 0,     -13182,0,    0,    0,    1014,
↦ 13182, 0,     -13182,0,     0,    0,    1014, 0,
↦ 0,     -13182,0,     0,     0,     1014, 0,     -1014,
↦ -13182,0,     0,     0,     1014, 0,     -1014,0,
↦ 0,     0,     0,     1014, 0,     -1014,0,     0,
↦ 0,     0,     1014, 0,     -1014,0,     0,     0,
↦ 0,     1014, 0,     -1014, 0,     0,     -338, 104,
↦ 1014, 0,     -1014, 0,     0,     0,     104, -26
```

See also: Section D.12.6.6 [matmult], page 2034; Section D.12.6.7 [tracemult], page 2034.

D.12.6.6 matmult

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: matmult(f,b,i); f poly, b,i ideal

Return: matrix: the matrix of the multiplication map by f (m_f) on r/i w.r.t. to the monomial basis b of r/i (r = basering)

Assume: i is a Groebner basis and b is an ordered monomial basis of r/i, as given by qbase(i)

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
poly f = x3-xy+y-13+x4-y2x;
matrix m = matmult(f,b,i);
print(m);
↦ 0,   1,  0,  -1,0,  0, 1,  0,
↦ 13,  0,  -13,0, 0,  0, 0,  1,
↦ 0,   0,  0,  1, 0,  -1,0,  0,
↦ 0,   0,  13, 0, -13,0, 0,  0,
↦ 0,   -13,0,  0, 0,  1, 0,  -1,
↦ -169,0,  0,  0, 13, 0, -13,0,
↦ 0,   0,  0,  0, 0,  0, -13,1,
↦ 0,   0,  0,  0, 0,  0, 13, -13
```

See also: Section D.12.6.8 [coords], page 2035; Section D.12.6.5 [matbil], page 2033.

D.12.6.7 tracemult

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: tracemult(f,B,I);f poly, B,I ideal

Return: number: the trace of the multiplication by f (m_f) on r/I, written in the monomial basis B of r/I, r = basering (faster than matmult + trace)

Assume: I is given by a Groebner basis and B is an ordered monomial basis of r/I

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
poly f = x3-xy+y-13+x4-y2x;
```

```
matrix m = matmult(f,b,i);
print(m);
↦ 0,   1,  0,  -1,0,  0, 1,  0,
↦ 13,  0,  -13,0, 0,  0, 0,  1,
↦ 0,   0,  0,  1, 0,  -1,0,  0,
↦ 0,   0,  13, 0, -13,0, 0,  0,
↦ 0,   -13,0,  0, 0,  1, 0,  -1,
↦ -169,0,  0,  0, 13, 0, -13,0,
↦ 0,   0,  0,  0, 0,  0, -13,1,
↦ 0,   0,  0,  0, 0,  0, 13, -13
tracemult(f,b,i);            //the trace of m
↦ -26
```

See also: Section D.12.6.6 [matmult], page 2034; Section 5.1.158 [trace], page 281.

D.12.6.8 coords

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: coords(f,b,i), f poly, b,i ideal

Return: list of numbers: the coordinates of the class of f (mod i) in the monomial basis b

Assume: i is a Groebner basis and b is an ordered monomial basis of r/i, r = basering

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
ideal i = x4-y2x,y2-13;
poly f = x3-xy+y-13+x4-y2x;
i = std(i);
ideal b = qbase(i);
b;
↦ b[1]=x3y
↦ b[2]=x3
↦ b[3]=x2y
↦ b[4]=x2
↦ b[5]=xy
↦ b[6]=x
↦ b[7]=y
↦ b[8]=1
coords(f,b,i);
↦ [1]:
↦ 0
↦ [2]:
↦ 1
↦ [3]:
↦ 0
↦ [4]:
↦ 0
↦ [5]:
↦ -1
↦ [6]:
↦ 0
↦ [7]:
↦ 1
↦ [8]:
↦ -13
```

See also: Section D.12.6.5 [matbil], page 2033; Section D.12.6.6 [matmult], page 2034.

D.12.6.9 randcharpoly

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: randcharpoly(b,i); randcharpoly(b,i,n); b,i ideal; n int

Return: poly: the characteristic polynomial of a pseudorandom rational univariate projection having one zero per zero of i. If n<10 is given, it is the number of digits being used for the pseudorandom coefficients (default: n=5)

Assume: i is a Groebner basis and b is an ordered monomial basis of r/i, r = basering

Note: shows a warning if printlevel>0 (default: printlevel=0)

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z),dp;
ideal i = (x-1)*(x-2),(y-1),(z-1)*(z-2)*(z-3)^2;
i = std(i);
ideal b = qbase(i);
poly p = randcharpoly(b,i);
p;
↦ z8-1989306z7+1720335326522z6-844575738768293508z5+257398574985069575973\
   5z4-49855396253842786126599566442z3+5991506449298102407845582886576172z2-\
   408335865183407651473343362162998177144z+120782617595757843238663349007\
   464660123776
nrroots(p); // See nrroots in urrcount.lib
↦ 6
int pr = printlevel;
printlevel = pr+2;
p = randcharpoly(b,i,5);
↦ // poly, 9 monomial(s)
↦ z8-2923964*z7+3712323518934*z6-2671920147197312780*z5+1191863249059288760\
   005489*z4-3372422352632042934615439390 56*z3+5907995204138272880895642574\
   100736*z2-5855367303472484622963975143953858560000*z+2512062931376195003\
   63959309186105344000000000
↦ *********************************************************************
↦ * WARNING: This polynomial was obtained using  pseudorandom numbers.*
↦ * If you want to verify the result, please use the command          *
↦ *                                                                   *
↦ * verify(p,b,i)                                                     *
↦ *                                                                   *
↦ * where p is the polynomial I returned, b is the monomial basis     *
↦ * used, and i the Groebner basis of the ideal                       *
↦ *********************************************************************
nrroots(p);
↦ 6
printlevel = pr;
```

D.12.6.10 verify

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: verify(p,B,I); p poly, B,I,ideal

Return: integer: 1 if and only if the polynomial p splits the points of V(I). It's used to check the result of randcharpoly

Assume: I is given by a Groebner basis and B is an ordered monomial basis of r/I, r = basering

Note: comments the result if printlevel>0 (default: printlevel=0)

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y),dp;
poly f = x3-xy+y-13+x4-y2x;
ideal i = x4-y2x,y2-13;
i = std(i);
ideal b = qbase(i);
poly p = randcharpoly(b,i);
verify(p,b,i);
↦ 1
```

See also: Section D.12.6.9 [randcharpoly], page 2036.

D.12.6.11 randlinpoly

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: randlinpoly(); randlinpoly(n); n int

Return: poly: linear combination of the variables of the ring, with pseudorandom coefficients. If n<10 is given, it is the number of digits being used for the range of the coefficients (default: n=5)

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z,w),dp;
poly p = randlinpoly();
p;
↦ 80035x+36642y+40875z+54263w
randlinpoly(5);
↦ 68857x+95664y+28174z+34170w
```

See also: Section D.12.6.9 [randcharpoly], page 2036.

D.12.6.12 powersums

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: powersums(f,b,i); f poly; b,i ideal

Return: list: the powersums of the results of evaluating f at the zeros of I

Assume: i is a Groebner basis and b is an ordered monomial basis of r/i, r = basering

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z),dp;
ideal i = (x-1)*(x-2),(y-1),(z+5); // V(I) = {(1,1,-5),(2,1,-5)}
i = std(i);
ideal b = qbase(i);
```

```
poly f = x+y+z;
list psums = list(-2-3,4+9); // f evaluated at V(I) gives {-3,-2}
list l = powersums(f,b,i);
psums;
↦ [1]:
↦    -5
↦ [2]:
↦    13
l;
↦ [1]:
↦ -5
↦ [2]:
↦ 13
```

See also: Section D.12.6.13 [symmfunc], page 2038.

D.12.6.13 symmfunc

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: symmfunc(s); s list

Return: list: the symmetric functions of the roots of a polynomial, given the power sums of those roots.

Example:

```
LIB "rootsmr.lib";
ring r = 0,x,dp;
poly p = (x-1)*(x-2)*(x-3);
list psums = list(1+2+3,1+4+9,1+8+27);
list l = symmfunc(psums);
l;
↦ [1]:
↦    1
↦ [2]:
↦ -6
↦ [3]:
↦ 11
↦ [4]:
↦ -6
p; // Compare p with the elements of l
↦ x3-6x2+11x-6
```

See also: Section D.12.6.12 [powersums], page 2037.

D.12.6.14 univarpoly

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: univarpoly(l); l list

Return: poly: a polynomial p on the first variable of basering, say x, with p = l[1] + l[2]*x + l[3]*x^2 + ...

Example:

```
LIB "rootsmr.lib";
ring r = 0,x,dp;
```

```
list l = list(1,2,3,4,5);
poly p = univarpoly(l);
p;
↦ x4+2x3+3x2+4x+5
```

D.12.6.15 qbase

Procedure from library `rootsmr.lib` (see Section D.12.6 [rootsmr_lib], page 2031).

Usage: qbase(I); I zero-dimensional ideal

Return: ideal: A monomial basis of the quotient between the basering and the ideal I, sorted according to the basering order.

Example:

```
LIB "rootsmr.lib";
ring r = 0,(x,y,z),dp;
ideal i = 2x2,-y2,z3;
i = std(i);
ideal b = qbase(i);
b;
↦ b[1]=xyz2
↦ b[2]=xyz
↦ b[3]=xz2
↦ b[4]=yz2
↦ b[5]=xy
↦ b[6]=xz
↦ b[7]=yz
↦ b[8]=z2
↦ b[9]=x
↦ b[10]=y
↦ b[11]=z
↦ b[12]=1
b = kbase(i);
b; // Compare this with the result of qbase
↦ b[1]=xyz2
↦ b[2]=yz2
↦ b[3]=xz2
↦ b[4]=z2
↦ b[5]=xyz
↦ b[6]=yz
↦ b[7]=xz
↦ b[8]=z
↦ b[9]=xy
↦ b[10]=y
↦ b[11]=x
↦ b[12]=1
```

See also: Section 5.1.69 [kbase], page 205.

D.12.7 rootsur_lib

Library: rootsur.lib

Purpose: Counting number of real roots of univariate polynomial

Author: Enrique A. Tobis, etobis@dc.uba.ar

Overview: Routines for bounding and counting the number of real roots of a univariate polynomial, by means of several different methods, namely Descartes' rule of signs, the Budan-Fourier theorem, Sturm sequences and Sturm-Habicht sequences. The first two give bounds on the number of roots. The other two compute the actual number of roots of the polynomial. There are several wrapper functions, to simplify the application of the aforesaid theorems and some functions to determine whether a given polynomial is univariate.

References:

Basu, Pollack, Roy, "Algorithms in Real Algebraic Geometry", Springer, 2003.

Procedures:

D.12.7.1 isuni

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: isuni(p); poly p;

Return: poly: if p is a univariate polynomial, it returns the variable. If not, zero.

Example:

```
LIB "rootsur.lib";
ring r = 0,(x,y),dp;
poly p = 6x7-3x2+2x-15/7;
isuni(p);
↦ x
isuni(p*y);
↦ 0
```

See also: Section D.12.7.2 [whichvariable], page 2040.

D.12.7.2 whichvariable

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: whichvariable(p); poly p

Return: poly: if p is a univariate monomial, the variable. Otherwise 0.

Assume: p is a monomial

Example:

```
LIB "rootsur.lib";
ring r = 0,(x,y),dp;
whichvariable(x5);
↦ x
whichvariable(x3y);
↦ 0
```

See also: Section D.12.7.1 [isuni], page 2040.

D.12.7.3 varsigns

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: varsigns(l); list l.

Return: int: the number of sign changes in the list l

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
list l = 1,2,3;
varsigns(l);
↦ 0
l = 1,-1,2,-2,3,-3;
varsigns(l);
↦ 5
```

See also: Section D.12.7.5 [boundposDes], page 2041.

D.12.7.4 boundBuFou

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: boundBuFou(p,a,b); p poly, a,b number

Return: int: an upper bound for the number of real roots of p in (a,b], with the same parity as the actual number of roots (using the Budan-Fourier Theorem)

Assume: - p is a univariate polynomial with rational coefficients
- a, b are rational numbers with a < b

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
boundBuFou(p,-3,5);
↦ 3
boundBuFou(p,-2,5);
↦ 2
```

See also: Section D.12.7.5 [boundposDes], page 2041; Section D.12.7.3 [varsigns], page 2041.

D.12.7.5 boundposDes

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: boundposDes(p); poly p

Return: int: an upper bound for the number of positive roots of p, with the same parity as the actual number of positive roots of p.

Assume: p is a univariate polynomial with rational coefficients

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
boundposDes(p);
```

```
↦ 2
p = p*(x2+1);
boundposDes(p);
↦ 4
```

See also: Section D.12.7.4 [boundBuFou], page 2041.

D.12.7.6 boundDes

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: boundDes(p); poly p

Return: int: an upper bound for the number of real roots of p, with the same parity as the actual number of real roots of p.

Assume: p is a univariate polynomial with rational coefficients

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
boundDes(p);
↦ 3
p = p*(x2+1);
boundDes(p);
↦ 5
```

See also: Section D.12.7.4 [boundBuFou], page 2041.

D.12.7.7 allrealst

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: allrealst(p); poly p

Return: int: 1 if and only if all the roots of p are real, 0 otherwise. Checks by using Sturm's Theorem whether all the roots of p are real

Assume: p is a univariate polynomial with rational coefficients

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
allrealst(p);
↦ 1
p = p*(x2+1);
allrealst(p);
↦ 0
```

See also: Section D.12.7.9 [allreal], page 2043; Section D.12.7.10 [sturm], page 2043; Section D.12.7.12 [sturmha], page 2044.

D.12.7.8 maxabs

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: maxabs(p); poly p

Return: number: an upper bound for the largest absolute value of a root of p

Assume: p is a univariate polynomial with rational coefficients

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
maxabs(p);
↦ 11
```

See also: Section D.12.7.10 [sturm], page 2043.

D.12.7.9 allreal

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: allreal(p);

Return: int: 1 if and only if all the roots of p are real, 0 otherwise

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
allreal(p);
↦ 1
p = p*(x2+1);
allreal(p);
↦ 0
```

See also: Section D.12.7.7 [allrealst], page 2042.

D.12.7.10 sturm

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: sturm(p,a,b); poly p, number a,b

Return: int: the number of real roots of p in (a,b]

Assume: p is a univariate polynomial with rational coefficients,
a, b are rational numbers with a < b

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
sturm(p,-3,6);
↦ 3
p = p*(x2+1);
sturm(p,-3,6);
↦ 3
```

```
p = p*(x+2);
sturm(p,-3,6);
↦ 3
```

See also: Section D.12.7.9 [allreal], page 2043; Section D.12.7.7 [allrealst], page 2042; Section D.12.7.12 [sturmha], page 2044.

D.12.7.11 sturmseq

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: sturmseq(p); p poly

Return: list: a Sturm sequence of p

Assume: p is a univariate polynomial with rational coefficients

Theory: The Sturm sequence of p (also called remainder sequence) is the sequence beginning with p, p' and goes on with the negative part of the remainder of the two previous polynomials, until the remainder is zero.
See: Basu, Pollack, Roy, Algorithms in Real Algebraic Geometry, Springer, 2003.

Example:

```
LIB "rootsur.lib";
ring r = 0,(z,x),dp;
poly p = x5-3x4+12x3+7x-153;
sturmseq(p);
↦ [1]:
↦    x5-3x4+12x3+7x-153
↦ [2]:
↦    x4-12/5x3+36/5x2+7/5
↦ [3]:
↦    -x3-9/7x2-5/3x+317/7
↦ [4]:
↦    -x2-756/151x+2433/151
↦ [5]:
↦    x-514191/177889
↦ [6]:
↦    1
```

See also: Section D.12.7.10 [sturm], page 2043; Section D.12.7.13 [sturmhaseq], page 2045.

D.12.7.12 sturmha

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: sturmha(p,a,b); poly p, number a,b

Return: int: the number of real roots of p in (a,b) (using a Sturm-Habicht sequence)

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
sturmha(p,-3,6);
↦ 3
p = p*(x2+1);
sturmha(p,-3,6);
```

```
↦ 3
```

See also: Section D.12.7.9 [allreal], page 2043; Section D.12.7.10 [sturm], page 2043.

D.12.7.13 sturmhaseq

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: sturmhaseq(P); P poly.

Return: list: the non-zero polynomials of the Sturm-Habicht sequence of P

Assume: P is a univariate polynomial.

Theory: The Sturm-Habicht sequence (also subresultant sequence) is closely related to the Sturm sequence, but behaves better with respect to the size of the coefficients. It is defined via subresultants. See: Basu, Pollack, Roy, Algorithms in Real Algebraic Geometry, Springer, 2003.

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = x5-x4+x-3/2;
list l = sturmhaseq(p);
l;
↦ [1]:
↦    132949/16
↦ [2]:
↦    -25x-332
↦ [3]:
↦    -16x2+42x-24
↦ [4]:
↦    4x3-20x+73/2
↦ [5]:
↦    5x4-4x3+1
↦ [6]:
↦    x5-x4+x-3/2
```

See also: Section D.12.7.10 [sturm], page 2043; Section D.12.7.12 [sturmha], page 2044; Section D.12.7.11 [sturmseq], page 2044.

D.12.7.14 reverse

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: reverse(l); l list

Return: list: l reversed.

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
list l = 1,2,3,4,5;
list rev = reverse(l);
l;
↦ [1]:
↦    1
↦ [2]:
```

```
↦    2
↦ [3]:
↦    3
↦ [4]:
↦    4
↦ [5]:
↦    5
rev;
↦ [1]:
↦    5
↦ [2]:
↦    4
↦ [3]:
↦    3
↦ [4]:
↦    2
↦ [5]:
↦    1
```

D.12.7.15 nrroots

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: nrroots(p); poly p

Return: int: the number of real roots of p

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
poly p = (x+2)*(x-1)*(x-5);
nrroots(p);
↦ 3
p = p*(x2+1);
nrroots(p);
↦ 3
```

See also: Section D.12.7.5 [boundposDes], page 2041; Section D.12.7.10 [sturm], page 2043; Section D.12.7.12 [sturmha], page 2044.

D.12.7.16 isparam

Procedure from library `rootsur.lib` (see Section D.12.7 [rootsur_lib], page 2039).

Usage: isparam(ideal/module/poly/list);

Return: int: 0 if the argument has non-parametric coefficients and 1 if it has parametric coefficients

Example:

```
LIB "rootsur.lib";
ring r = 0,x,dp;
isparam(2x3-56x+2);
↦ 0
ring s = (0,a,b,c),x,dp;
isparam(2x3-56x+2);
```

```
↦ 0
isparam(2x3-56x+abc);
↦ 1
```

D.13 Tropical Geometry

D.13.1 cimonom_lib

Library: cimonom.lib

Purpose: Determines if the toric ideal of an affine monomial curve is a complete intersection

Authors: I.Bermejo, ibermejo@ull.es
I.Garcia-Marco, iggarcia@ull.es
J.-J.Salazar-Gonzalez, jjsalaza@ull.es

Overview: A library for determining if the toric ideal of an affine monomial curve is a complete intersection with NO NEED of computing explicitly a system of generators of such ideal. It also contains procedures to obtain the minimum positive multiple of an integer which is in a semigroup of positive integers. The procedures are based on a paper by Isabel Bermejo, Ignacio Garcia and Juan Jose Salazar-Gonzalez: 'An algorithm to check whether the toric ideal of an affine monomial curve is a complete intersection', Preprint.

Procedures: See also: Section C.6.4 [Integer programming], page 783.

D.13.1.1 BelongSemig

Procedure from library `cimonom.lib` (see Section D.13.1 [cimonom_lib], page 2047).

Usage: BelongSemig (n,v[,sup]); n bigint, v and sup intvec

Return: In the default form, it returns 1 if n is in the semigroup generated by the elements of v or 0 otherwise. If the argument sup is added and in case n belongs to the semigroup generated by the elements of v, it returns a monomial in the variables {x(i) | i in sup} of degree n if we set deg(x(sup[j])) = v[j].

Assume: v and sup positive integer vectors of same size, sup has no repeated entries, x(i) has to be an indeterminate in the current ring for all i in sup.

Example:

```
LIB "cimonom.lib";
ring r=0,x(1..5),dp;
int a = 125;
intvec v = 13,17,51;
intvec sup = 2,4,1;
BelongSemig(a,v,sup);
↦ x(2)^7*x(4)^2
BelongSemig(a,v);
↦ 1
```

D.13.1.2 MinMult

Procedure from library `cimonom.lib` (see Section D.13.1 [cimonom_lib], page 2047).

Usage: MinMult (a, b); a integer, b integer vector.

Return: an integer k, the minimum positive integer such that ka belongs to the semigroup generated by the integers in b.

Assume: a is a positive integer, b is a positive integers vector.

Example:

```
LIB "cimonom.lib";
"int a = 46;";
↦ int a = 46;
"intvec b = 13,17,59;";
↦ intvec b = 13,17,59;
"MinMult(a,b);";
↦ MinMult(a,b);
int a = 46;
intvec b = 13,17,59;
MinMult(a,b);
↦ 3
"// 3*a = 8*b[1] + 2*b[2]"
```

D.13.1.3 CompInt

Procedure from library `cimonom.lib` (see Section D.13.1 [cimonom_lib], page 2047).

Usage: CompInt(d); d intvec.

Return: 1 if the toric ideal I(d) is a complete intersection or 0 otherwise.

Assume: d is a vector of positive integers.

Note: If printlevel > 0, additional info is displayed in case I(d) is a complete intersection: if printlevel >= 1, it displays a minimal set of generators of the toric ideal formed by quasihomogeneous binomials. Moreover, if printlevel >= 2 and gcd(d) = 1, it also shows the Frobenius number of the semigroup generated by the elements in d.

Example:

```
LIB "cimonom.lib";
printlevel = 0;
intvec d = 14,15,10,21;
CompInt(d);
↦ 1
printlevel = 3;
d = 36,54,125,150,225;
CompInt(d);
↦ // Toric ideal:
↦ id[1]=-x(1)^3+x(2)^2
↦ id[2]=-x(4)^3+x(5)^2
↦ id[3]=-x(3)^3+x(4)*x(5)
↦ id[4]=-x(1)^11*x(2)+x(4)^3
↦ // Frobenius number of the numerical semigroup:
↦ 793
↦ 1
```

```
d = 45,70,75,98,147;
CompInt(d);
↦ 0
```

D.13.2 gfan_lib

Library: gfan.lib

Purpose: Interface to gfan and gfanlib for computations in convex geometry

Authors: Anders N. Jensen, email: jensen@imf.au.dk
Yue Ren, email: ren@mathematik.uni-kl.de
Frank Seelisch

Procedures:

D.13.2.1 fullSpace

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: fullSpace(n); n int

Return: cone, the ambient space of dimension n

Example:

```
LIB "gfan.lib";
cone c = fullSpace(2);
generatorsOfLinealitySpace(c);
↦ -1, 0,
↦  0,-1
```

D.13.2.2 origin

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: origin(n); n int

Return: cone, the origin in an ambient space of dimension n

Example:

```
LIB "gfan.lib";
cone c = origin(2);
equations(c);
↦ 1,0,
↦ 0,1
```

D.13.2.3 positiveOrthant

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: positiveOrthant(n); n int

Return: cone, the positive orthant of dimension n

Example:

```
LIB "gfan.lib";
cone c = positiveOrthant(2);
rays(c);
↦ 1,0,
↦ 0,1
```

D.13.2.4 ambientDimension

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: ambientDimension(c); c cone
ambientDimension(f); f fan
ambientDimension(p); p polytope

Return: int, the dimension of the ambient space the input lives in

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
ambientDimension(c1);
↦ 2
intmat M2[2][3]=
1,0,0,
0,1,0;
cone c2=coneViaPoints(M2);
ambientDimension(c2);
↦ 3
fan f = emptyFan(3);
ambientDimension(f);
↦ 3
```

D.13.2.5 canonicalizeCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: canonicalizeCone(c); c cone

Return: cone, a unique representation of the cone c

Example:

```
LIB "gfan.lib";
intmat M[5][3]=
8,1,9,
9,2,4,
0,6,2,
8,8,8,
0,9,5;
cone c=coneViaInequalities(M);
c;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦ 0,3,1,
↦ 0,9,5,
↦ 1,1,1,
↦ 8,1,9,
↦ 9,2,4
↦ LINEAR_SPAN
↦
```

```
↦
cone cc=canonicalizeCone(c);
cc;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,3,1,
↦ 0,9,5,
↦ 8,1,9,
↦ 9,2,4
↦ LINEAR_SPAN
↦
↦
// computes a unique representation of c
c == cc;
↦ 1
// some procedures work with the known inequalities and equations
// in order to obtain a unique output,
// bring the cone in canonical form beforehand
relativeInteriorPoint(c);
↦ 7,4,-6
relativeInteriorPoint(cc);
↦ 7,4,-6
```

D.13.2.6 codimension

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: codimension(c); c cone
codimension(f); f fan
codimension(p); p polytope

Return: int, the codimension of the input

Example:

```
LIB "gfan.lib";
intmat M1[1][2]=
1,0;
cone c1=coneViaPoints(M1);
codimension(c1);
↦ 1
intmat M2[2][2]=
1,0,
0,1;
cone c2=coneViaPoints(M2);
codimension(c2);
↦ 0
fan f = emptyFan(2);
codimension(f);
↦ -1
insertCone(f,c1);
codimension(f);
↦ 1
insertCone(f,c2);
```

```
codimension(f);
↦ 0
```

D.13.2.7 coneViaPoints

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: coneViaPoints(HL); intmat HL
coneViaPoints(HL,L); intmat HL, intmat L
coneViaPoints(HL,L,flags); intmat HL, intmat L, int flags

Return: cone

Purpose: cone generated by half lines generated by the row vectors of HL and (if stated) by lines generated by the row vectors of L; flags may range between 0,..,3 defining an upper and lower bit (0=0*2+0, 1=0*2+1, 2=1*2+0, 3=1*2+1),
if upper bit is 1, then program assumes that each row vector in HL generates a ray of the cone,
if lower bit is 1, then program assumes that the span of the row vectors of L is the lineality space of the cone,
if either bit is 0, then program computes the information itself.

Example:

```
LIB "gfan.lib";
// Let's define a cone in R^3 generated by the following half lines:
intmat HL[5][3]=
1,0, 0,
-1,0, 0,
0,1, 1,
0,1,-1,
0,0, 1;
cone c=coneViaPoints(HL);
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,0,
↦ 0,1,1
↦ LINEAR_SPAN
↦
↦
kill HL,c;
// Note that (1,0,0) and (-1,0,0) form a line, hence also possible:
intmat HL[3][3]=
0,1, 1,
0,1,-1,
0,0, 1;
intmat L[1][3]=
1,0,0;
cone c=coneViaPoints(HL,L);
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,0,
↦ 0,1,1
↦ LINEAR_SPAN
↦
↦
kill HL,L,c;
// lineality space is exactly Lin(1,0,0)
intmat HL[3][3]=
0,1, 1,
0,1,-1,
0,0, 1;
intmat L[1][3]=
1,0,0;
cone c=coneViaPoints(HL,L,1);
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,0,
↦ 0,1,1
↦ LINEAR_SPAN
↦
↦
kill HL,L,c;
// and that (0,1,-1), (0,1,1) generate rays
intmat HL[3][3]=
0,1, 1,
0,1,-1;
intmat L[1][3]=
1,0,0;
cone c=coneViaPoints(HL,L,1);
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,-1,
↦ 0,1, 1
↦ LINEAR_SPAN
↦
↦
kill HL,L,c;
// and that (0,1,-1), (0,1,1) generate rays
intmat HL[3][3]=
0,1, 1,
0,1,-1;
intmat L[1][3]=
1,0,0;
cone c=coneViaPoints(HL,L,3);
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,-1,
↦ 0,1, 1
↦ LINEAR_SPAN
↦
↦
```

D.13.2.8 coneViaInequalities

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: coneViaInequalities(IE); intmat IE
coneViaInequalities(IE,E); intmat IE, intmat E
coneViaInequalities(IE,E,flags); intmat IE, intmat E, int flags

Return: cone

Purpose: cone consisting of all points x, such that IE*x >= 0 in each component and (if stated) E*x = 0;
inequalities and (if stated) equations will be transformed, getting rid of redundancies;
flags may range between 0,..,3 defining an upper and lower bit (0=0*2+0, 1=0*2+1, 2=1*2+0, 3=1*2+1),
if higher bit is 1, then program assumes each inequality yields a facet, if lower bit is 1, then program assumes the kernel of E is the span of the cone, if either bit is 0, then program computes the information itself.

Example:

```
LIB "gfan.lib";
// Let's define a cone in R^3 given by the following inequalities:
intmat IE[6][3]=
1,3,5,
1,5,3,
0,1,-1,
0,1,1,
1,0,0,
-1,0,0;
cone c=coneViaInequalities(IE);
c;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦ 0,1,-1,
↦ 0,1, 1,
↦ 1,3, 5,
↦ 1,5, 3
↦ LINEAR_SPAN
↦ -1,0,0
↦
// Note that the last two inequalities yield x1 = 0, hence also possible:
intmat IE[4][3]=
↦ // ** redefining IE (intmat IE[4][3]=) ./examples/coneViaInequalities.sin\
   g:13
0,1,-1,
0,1,1;
intmat E[1][3]=
1,0,0;
```

```
cone c=coneViaInequalities(IE,E);
↦ // ** redefining c (cone c=coneViaInequalities(IE,E);) ./examples/coneVia\
   Inequalities.sing:18
c;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦ 0,1,-1,
↦ 0,1, 1
↦ LINEAR_SPAN
↦ 1,0,0
↦ 
// each inequalities gives rise to a facet
intmat IE[2][3]=
↦ // ** redefining IE (intmat IE[2][3]=) ./examples/coneViaInequalities.sin\
   g:21
0,1,-1,
0,1,1;
intmat E[1][3]=
↦ // ** redefining E (intmat E[1][3]=) ./examples/coneViaInequalities.sing:\
   24
1,0,0;
cone c=coneViaInequalities(IE,E,1);
↦ // ** redefining c (cone c=coneViaInequalities(IE,E,1);) ./examples/coneV\
   iaInequalities.sing:26
c;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦ 0,1,-1,
↦ 0,1, 1
↦ LINEAR_SPAN
↦ 1,0,0
↦ 
// and the kernel of E is the span of the cone
intmat IE[2][3]=
↦ // ** redefining IE (intmat IE[2][3]=) ./examples/coneViaInequalities.sin\
   g:29
0,1,-1,
0,1,1;
intmat E[1][3]=
↦ // ** redefining E (intmat E[1][3]=) ./examples/coneViaInequalities.sing:\
   32
1,0,0;
cone c=coneViaInequalities(IE,E,3);
↦ // ** redefining c (cone c=coneViaInequalities(IE,E,3);) ./examples/coneV\
   iaInequalities.sing:34
c;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,1,-1,
↦ 0,1, 1
↦ LINEAR_SPAN
↦ 1,0,0
↦
```

D.13.2.9 coneLink

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: coneLink(c,w); c cone, w intvec/bigintmat

Return: cone, the link of c around w

Example:

```
LIB "gfan.lib";
intmat M[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c=coneViaPoints(M);
intvec v=1,0,0;
cone cv=coneLink(c,v);
rays(cv);
↦ 0,1,0,
↦ 0,0,1
generatorsOfLinealitySpace(cv);
↦ -1,0,0
intvec w=1,1,1;
cone cw=coneLink(c,w);
rays(cw);
↦
generatorsOfLinealitySpace(cw);
↦ -1, 0, 0,
↦  0,-1, 0,
↦  0, 0,-1
```

D.13.2.10 containsAsFace

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: containsAsFace(c,d); c cone, d cone

Return: 1, if d is a face of c; 0 otherwise

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
intmat N1[1][2]=
1,1;
cone d1=coneViaPoints(N1);
containsInSupport(c,d1);
↦ 1
containsAsFace(c,d1);
↦ 0
```

```
intmat N2[1][2]=
0,1;
cone d2=coneViaPoints(N2);
containsInSupport(c,d2);
↦ 1
containsAsFace(c,d2);
↦ 1
```

D.13.2.11 containsInSupport

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: containsInSupport(c,d); c cone, d cone
containsInSupport(c,p); c cone, p intvec/bigintmat

Return: 1, if d resp. p is contained in c; 0 otherwise

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
containsInSupport(c,c);
↦ 1
intmat N1[2][2]=
1,1,
0,1;
cone d1=coneViaPoints(N1);
containsInSupport(c,d1);
↦ 1
intmat N2[2][2]=
1,1,
1,-1;
cone d2=coneViaPoints(N2);
containsInSupport(c,d2);
↦ 0
intvec p1=0,1;
containsInSupport(c,p1);
↦ 1
intvec p2=1,-1;
containsInSupport(c,p2);
↦ 0
```

D.13.2.12 containsPositiveVector

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: containsPositiveVector(c); c cone

Return: 1, if c contains a vector with only positive entries in its relative interior

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,1,
1,-1;
cone c1=coneViaPoints(M1);
containsPositiveVector(c1);
↦ 1
intmat M2[2][2]=
0,1,
-1,0;
cone c2=coneViaPoints(M2);
containsPositiveVector(c2);
↦ 0
```

D.13.2.13 containsRelatively

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: containsRelatively(c,p); c cone, intvec p

Return: 1 iff the given cone contains the given point in its relative interior; 0 otherwise

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
intvec p1=1,1;
containsRelatively(c,p1);
↦ 1
intvec p2=0,1;
containsRelatively(c,p2);
↦ 0
```

D.13.2.14 convexHull

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: convexHull(c1,c2); c1 cone, c2 cone
convexHull(c1,p1); c1 cone, p1 polytope
convexHull(p1,c1); p1 cone, c1 polytope
convexHull(p1,p2); p1 polytope, p2 polytope

Return: cone resp polytope, the convex hull of its two input objects

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
intmat M2[2][2]=
1,1,
1,-1;
cone c2=coneViaPoints(M2);
intmat M3[2][2]=
1,0,
0,-1;
cone c3=coneViaPoints(M3);
cone c12=convexHull(c1,c2);
c12;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 1,0,
↦ 1,1
↦ LINEAR_SPAN
↦
↦
print(rays(c12));
↦ 1,-1,
↦ 0, 1
cone c23=convexHull(c2,c3);
c23;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 1,-1,
↦ 1, 0
↦ LINEAR_SPAN
↦
↦
print(rays(c23));
↦ 0,-1,
↦ 1, 1
cone c13=convexHull(c1,c3);
c13;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 1,0
↦ LINEAR_SPAN
↦
↦
print(rays(c13));
↦ 1,0
```

D.13.2.15 convexIntersection

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: convexIntersection(c1,c2); c1 cone, c2 cone
convexIntersection(c1,p1); c1 cone, p1 polytope
convexIntersection(p1,c1); p1 cone, c1 polytope
convexIntersection(p1,p2); p1 polytope, p2 polytope

Return: cone resp polytope, the convex hull of its two input objects

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
intmat M2[2][2]=
1,1,
1,-1;
cone c2=coneViaPoints(M2);
intmat M3[2][2]=
1,0,
0,-1;
cone c3=coneViaPoints(M3);
cone c12=convexIntersection(c1,c2);
c12;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 0, 1,
↦ 1,-1
↦ LINEAR_SPAN
↦
↦
print(rays(c12));
↦ 1,0,
↦ 1,1
cone c23=convexIntersection(c2,c3);
c23;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 0,-1,
↦ 1, 1
↦ LINEAR_SPAN
↦
↦
print(rays(c23));
↦ 1, 0,
↦ 1,-1
cone c13=convexIntersection(c1,c3);
c13;
↦ AMBIENT_DIM
↦ 2
↦ FACETS
↦ 1,0
↦ LINEAR_SPAN
↦ 0,1
↦
print(rays(c13));
↦ 1,0
```

D.13.2.16 dimension

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: dimension(c); c cone
dimension(f); f fan
dimension(p); p polytope

Return: int, the dimension of the input

Example:

```
LIB "gfan.lib";
intmat M1[1][2]=
1,0;
cone c1=coneViaPoints(M1);
dimension(c1);
↦ 1
intmat M2[2][2]=
1,0,
0,1;
cone c2=coneViaPoints(M2);
dimension(c2);
↦ 2
fan f = emptyFan(2);
dimension(f);
↦ -1
insertCone(f,c1);
dimension(f);
↦ 1
insertCone(f,c2);
dimension(f);
↦ 2
```

D.13.2.17 dualCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: dualCone(c); c cone

Return: cone, the dual of c

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
cone d1=dualCone(c1);
d1;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,1,
↦ 1,0
↦ LINEAR_SPAN
↦
↦
print(rays(d1));
↦ 1,0,
↦ 0,1
intmat M2[2][2]=
1,1,
0,1;
cone c2=coneViaPoints(M2);
cone d2=dualCone(c2);
d2;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,1,
↦ 1,1
↦ LINEAR_SPAN
↦
↦
print(rays(d2));
↦  1,0,
↦ -1,1
```

D.13.2.18 equations

Procedure from library **gfan.lib** (see Section D.13.2 [gfan_lib], page 2049).

Usage: equations(c); c cone
equations(p); p polytope

Return: bigintmat, defining equations of c resp p

Note: neither unique nor complete, unless c resp p in canonical form

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
bigintmat E1=equations(c1);
print(E1);
↦
intmat M2[1][2]=
1,0;
cone c2=coneViaPoints(M2);
bigintmat E2=equations(c2);
print(E2);
↦ 0,-1
```

D.13.2.19 faceContaining

Procedure from library **gfan.lib** (see Section D.13.2 [gfan_lib], page 2049).

Usage: faceContaining(c,w); c cone, w intvec/bigintmat

Assume: containsInSupport(c,w)==1

Return: cone, the face of c containing w in its relative interior

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
faceContaining(c,intvec(1,0));
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 1,0
↦ LINEAR_SPAN
↦ 0,1
↦
faceContaining(c,intvec(0,1));
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,1
↦ LINEAR_SPAN
↦ 1,0
↦
faceContaining(c,intvec(1,1));
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,1,
↦ 1,0
↦ LINEAR_SPAN
↦
↦
faceContaining(c,intvec(0,0));
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦
↦ LINEAR_SPAN
↦ 1,0,
↦ 0,1
↦
```

D.13.2.20 facets

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: facets(c); c cone
facets(p); p polytope

Return: bigintmat, the facets of c resp p

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
```

```
cone c1=coneViaPoints(M1);
bigintmat F1=facets(c1);
print(F1);
↦ 0,1,
↦ 1,0
intmat M2[2][2]=
1,1,
0,-1;
cone c2=coneViaPoints(M2);
bigintmat F2=facets(c2);
print(F2);
↦ 1,-1,
↦ 1, 0
```

D.13.2.21 generatorsOfLinealitySpace

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: generatorsOfLinealitySpace(c); c cone

Return: bigintmat, generators of the lineality space of c

Example:

```
LIB "gfan.lib";
intmat M[5][3]=
1,0,0,
0,1,0,
0,0,1,
-1,0,0,
0,-1,0;
cone c=coneViaPoints(M);
bigintmat L=generatorsOfLinealitySpace(c);
print(L);
↦ -1, 0,0,
↦  0,-1,0
```

D.13.2.22 generatorsOfSpan

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: generatorsOfSpan(c); c cone

Return: bigintmat, generators of the span of c

Example:

```
LIB "gfan.lib";
intmat M[3][5]=
1,0,0,0,0,
0,1,0,0,0,
0,0,1,0,0;
cone c=coneViaPoints(M);
bigintmat S=generatorsOfSpan(c);
print(S);
↦ -1, 0, 0,0,0,
↦  0,-1, 0,0,0,
↦  0, 0,-1,0,0
```

D.13.2.23 getLinearForms

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: getLinearForms(c); c cone
getLinearForms(p); p polytope

Return: bigintmat, linear forms previously stored in c resp p

Example:

```
LIB "gfan.lib";
intmat M[2][3]=
-1,0,0,
0,-1,0;
cone c=coneViaPoints(M);
getLinearForms(c);
↦
intvec v=1,1,1;
setLinearForms(c,v);
getLinearForms(c);
↦ 1,1,1
```

D.13.2.24 getMultiplicity

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: getMultiplicity(c); c cone
getMultiplicity(p); p polytope

Return: bigint, 1 or a multiplicity previously stored in c resp p

Example:

```
LIB "gfan.lib";
intmat M[2][3]=
-1,0,0,
0,-1,0;
cone c=coneViaPoints(M);
getMultiplicity(c);
↦ 1
setMultiplicity(c,3);
getMultiplicity(c);
↦ 3
```

D.13.2.25 inequalities

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: inequalities(c); c cone
inequalities(p); p polytope

Return: bigintmat, the inequalities of c resp p

Note: neither unique nor irredundant, unless c resp p in canonical form

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
bigintmat I1=inequalities(c1);
print(I1);
↦ 0,1,
↦ 1,0
intmat M2[2][2]=
1,1,
0,-1;
cone c2=coneViaPoints(M2);
bigintmat I2=inequalities(c2);
print(I2);
↦ 1,-1,
↦ 1, 0
```

D.13.2.26 isFullSpace

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: isFullSpace(c); c cone

Return: 1, if c is the entire ambient space; 0 otherwise

Example:

```
LIB "gfan.lib";
cone c1;
isFullSpace(c1);
↦ 1
intmat M2[2][2]=
1,0,
0,1;
cone c2=coneViaPoints(M2);
isFullSpace(c2);
↦ 0
intmat M3[4][2]=
1,0,
0,1,
-1,0,
0,-1;
cone c3=coneViaPoints(M3);
isFullSpace(c3);
↦ 1
```

D.13.2.27 isOrigin

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: isOrigin(c); c cone

Return: 1, if c is the origin; 0 otherwise

Example:

```
LIB "gfan.lib";
cone c1;
isOrigin(c1);
↦ 1
intmat M2[2][2]=
1,0,
0,1;
cone c2=coneViaPoints(M2);
isOrigin(c2);
↦ 0
intmat M3[4][2]=
1,0,
0,1,
-1,0,
0,-1;
cone c3=coneViaPoints(M3);
isOrigin(c3);
↦ 0
```

D.13.2.28 isSimplicial

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: isSimplicial(c); c cone
isSimplicial(f); f fan

Return: 1, if c resp f is simplicial; 0 otherwise

Example:

```
LIB "gfan.lib";
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
isSimplicial(c1);
↦ 1
intmat M2[4][3]=
1,0,0,
0,1,0,
0,0,1,
1,1,-1;
cone c2=coneViaPoints(M2);
isSimplicial(c2);
↦ 0
/***********************/
fan f=emptyFan(3);
isSimplicial(f);
↦ 1
intmat N1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone d1=coneViaPoints(N1);
```

```
insertCone(f,d1);
isSimplicial(f);
↦ 1
intmat N2[4][3]=
1,0,0,
0,1,0,
1,0,-1,
0,1,-1;
cone d2=coneViaPoints(N2);
insertCone(f,d2);
isSimplicial(f);
↦ 0
```

D.13.2.29 linealityDimension

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: linealityDimension(c); c cone
linealityDimension(f); f fan

Return: int, the dimension of the lineality space of c resp f

Example:

```
LIB "gfan.lib";
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
linealityDimension(c1);
↦ 0
intmat M2[4][3]=
1,0,0,
0,1,0,
0,0,1,
-1,0,0;
cone c2=coneViaPoints(M2);
linealityDimension(c2);
↦ 1
```

D.13.2.30 linealitySpace

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: linealitySpace(c); c cone

Return: cone, the lineality space of c

Example:

```
LIB "gfan.lib";
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
cone l1=linealitySpace(c1);
```

```
l1;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦
↦ LINEAR_SPAN
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
↦
intmat M2[4][3]=
1,0,0,
0,1,0,
0,0,1,
-1,0,0;
cone c2=coneViaPoints(M2);
cone l2=linealitySpace(c2);
l2;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦
↦ LINEAR_SPAN
↦ 0,1,0,
↦ 0,0,1
↦
```

D.13.2.31 negatedCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: negatedCone(c); c cone

Return: cone, the negative of c

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
cone cn=negatedCone(c);
print(rays(cn));
↦ -1, 0,
↦  0,-1
```

D.13.2.32 polytopeViaInequalities

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: polytopeViaInequalities(EV [, E [, flags]]); intmat EV,E, int flags

Return: polytope consisting of all points x, such that IE*x >= 0 in each component and (if stated) E*x = 0;
flags may range between 0,..,3 defining an upper and lower bit (0=0*2+0, 1=0*2+1,

2=1*2+0, 3=1*2+1),
if higher bit is 1, then program assumes each inequality yields a facet, if lower bit is 1, then program assumes the kernel of E is the span of the cone, if either bit is 0, then program computes the information itself.

Example:

```
LIB "gfan.lib";
intmat IE[2][3]=
1,0,0,
0,1,0;
intmat E[1][3]=
0,0,1;
polytope p=polytopeViaInequalities(IE,E);
p;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,1,0,
↦ 1,0,0
↦ EQUATIONS
↦ 0,0,1
↦
```

D.13.2.33 polytopeViaPoints

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: polytopeViaPoints(V [, flags]); intmat V, int flags

Return: polytope which is the intersection of the cone generated by the row vectors of V with the hyperplane, in which the first coordinate equals 1; flags may be 0 or 1,
if flags is 1, then program assumes that each row vector of M generates a ray in the cone, if flags is 0, then program computes that information itself

Example:

```
LIB "gfan.lib";
// This is a polytope in R^2 generated by (0,0), (1,0), (0,1), (0,0);
intmat V[4][3]=
1,0,0,
1,1,0,
1,0,1,
1,1,1;
polytope p1=polytopeViaPoints(V);
p1;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0, 0, 1,
↦ 0, 1, 0,
↦ 1,-1, 0,
↦ 1, 0,-1
↦ EQUATIONS
↦
↦
```

```
// This is a polytope in R^2 generated by (1/2,2/3), (3/4,4/5), (5/6,6/7):
intmat V[3][3]=
↦ // ** redefining V (intmat V[3][3]=) ./examples/polytopeViaPoints.sing:11
6,3,4,
20,15,16,
42,35,36;
polytope p2=polytopeViaPoints(V);
p2;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ -10,-24, 35,
↦  -6, -8, 15,
↦   8, 12,-21
↦ EQUATIONS
↦
↦
// This polytope is the positive orthant in R^2:
// (0,1,0) and (0,0,1) imply that the polytope is unbounded in that direction
intmat V[3][3]=
↦ // ** redefining V (intmat V[3][3]=) ./examples/polytopeViaPoints.sing:19
1,0,0,
0,1,0,
0,0,1;
polytope p3=polytopeViaPoints(V);
p3;
↦ AMBIENT_DIM
↦ 2
↦ INEQUALITIES
↦ 0,0,1,
↦ 0,1,0,
↦ 1,0,0
↦ EQUATIONS
↦
↦
```

D.13.2.34 quotientLatticeBasis

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: quotientLatticeBasis(c); c cone

Return: bigintmat, a basis of Z^n intersected with the span of c modulo Z^n intersected with the lineality space of c

Example:

```
LIB "gfan.lib";
intmat M[3][2]=
1,0,
0,1,
-1,0;
cone c=coneViaPoints(M);
bigintmat Q=quotientLatticeBasis(c);
print(Q);
↦ 0,1
```

D.13.2.35 randomPoint

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: randomPoint(c); c cone
randomPoint(c,b); c cone, b int

Return: bigintmat, a random point in the relative interior of c

Note: returns a weighted sum over all its rays
if b is given and b>0, only chooses weights between 1 and b

Example:

```
LIB "gfan.lib";
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
bigintmat Q=randomPoint(c);
print(Q);
↦ 1335380034,380636641
bigintmat P=randomPoint(c,5);
print(P);
↦ 4,4
```

D.13.2.36 rays

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: rays(c); c cone

Return: bigintmat, generators of the rays of c, orthogonal to its lineality space

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
bigintmat R1=rays(c1);
print(R1);
↦ 1,0,
↦ 0,1
intmat M2[3][2]=
1,0,
0,1,
-1,0;
cone c2=coneViaPoints(M2);
bigintmat R2=rays(c2);
print(R2);
↦ 0,1
```

D.13.2.37 relativeInteriorPoint

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: relativeInteriorPoint(c); c cone

Return: bigintmat, a point in the relative interior of c

Note: not unique, unless c is in its canonical form

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
relativeInteriorPoint(c1);
↦ 1,1
intmat M2[2][2]=
1,0,
1,1;
cone c2=coneViaPoints(M2);
relativeInteriorPoint(c2);
↦ 2,1
```

D.13.2.38 semigroupGenerator

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: semigroupGenerator(c); c cone

Return: bigintmat, the generator of Z^n intersected with c modulo Z^n intersected with the lineality space of c

Assume: dimension(c) == linealityDimension(c)+1

Example:

```
LIB "gfan.lib";
intmat M[3][2]=
1,0,
0,1,
-1,0;
cone c=coneViaPoints(M);
semigroupGenerator(c);
↦ 0,1
```

D.13.2.39 setLinearForms

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: setLinearForms(c); c cone
setLinearForms(p); p polytope

Return: none, stores linear forms in c resp p

Example:

```
LIB "gfan.lib";
intmat M[2][3]=
-1,0,0,
0,-1,0;
cone c=coneViaPoints(M);
getLinearForms(c);
↦
```

```
intvec v=1,1,1;
setLinearForms(c,v);
getLinearForms(c);
↦ 1,1,1
```

D.13.2.40 setMultiplicity

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: setMultiplicity(c); c cone
setMultiplicity(p); p polytope

Return: none, stores a multiplicity in c resp p

Example:

```
LIB "gfan.lib";
intmat M[2][3]=
-1,0,0,
0,-1,0;
cone c=coneViaPoints(M);
getMultiplicity(c);
↦ 1
setMultiplicity(c,3);
getMultiplicity(c);
↦ 3
```

D.13.2.41 span

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: span(c); c cone

Return: bigintmat, unique irredundant equations of c

Note: the name 'span' was chosen to be in line with polymake's nomenclature

Example:

```
LIB "gfan.lib";
intmat M[3][5]=
1,0,0,0,0,
0,1,0,0,0,
0,0,1,0,0;
cone c=coneViaPoints(M);
bigintmat Eq=span(c);
print(Eq);
↦ 0,0,0,-1, 0,
↦ 0,0,0, 0,-1
```

D.13.2.42 uniquePoint

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: uniquePoint(c); c cone

Return: bigintmat, a unique point in c stable under reflections at coordinate hyperplanes

Example:

```
LIB "gfan.lib";
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
uniquePoint(c1);
↦ 1,1
intmat M2[2][2]=
-1,0,
0,1;
cone c2=coneViaPoints(M2);
uniquePoint(c2);
↦ -1,1
```

D.13.2.43 containsInCollection

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: containsInCollection(f,c); f fan, c cone

Return: 1, if f contains c; 0 otherwise

Example:

```
LIB "gfan.lib";
fan f=emptyFan(2);
intmat M[2][2]=
1,0,
0,1;
cone c=coneViaPoints(M);
containsInCollection(f,c);
↦ 0
insertCone(f,c);
containsInCollection(f,c);
↦ 1
```

D.13.2.44 emptyFan

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: emptyFan(n); n int

Return: fan, an empty fan in ambient dimension n

Example:

```
LIB "gfan.lib";
fan f=emptyFan(2);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ -1
```

```
↦
↦ LINEALITY_DIM
↦ 2
↦
↦ RAYS
↦
↦ N_RAYS
↦ 0
↦
↦ LINEALITY_SPACE
↦ 1 0 # 0
↦ 0 1 # 1
↦
↦ ORTH_LINEALITY_SPACE
↦
↦ F_VECTOR
↦
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦
↦ MAXIMAL_CONES
↦
```

D.13.2.45 fanViaCones

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: fanViaCones(L); L list
fanViaCones(c1[,...,ck]); c1,...,ck cones

Return: fan, creates a fan generated by the cones in L resp c1,...,ck

Example:

```
LIB "gfan.lib";
intmat M[2][2]=1,0,0,1;
cone c=coneViaPoints(M);
intmat N[2][2]=1,0,0,-1;
cone d=coneViaPoints(N);
fan f=fanViaCones(c,d);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 0 -1 # 0
↦ 0 1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 2
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 2} # Dimension 2
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 2} # Dimension 2
↦ {1 2}
↦
list L=c,d;
fan g=fanViaCones(L);
g;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
```

```
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 0 -1 # 0
↦ 0 1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 2
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 2} # Dimension 2
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 2} # Dimension 2
↦ {1 2}
↦
```

D.13.2.46 fullFan

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: fullFan(n); n int

Return: fan, an full fan in ambient dimension n

Example:

```
LIB "gfan.lib";
fan f=fullFan(2);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
```

```
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 2
↦
↦ RAYS
↦
↦ N_RAYS
↦ 0
↦
↦ LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ ORTH_LINEALITY_SPACE
↦
↦ F_VECTOR
↦ 1
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 2
↦
↦ MAXIMAL_CONES
↦ {} # Dimension 2
↦
```

D.13.2.47 fVector

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: fVector(f); f fan

Return: bigintmat, the f-Vector of f

Example:

```
LIB "gfan.lib";
fan f=emptyFan(2);
fVector(f);
↦
intmat M[2][2]=1,0,0,1;
cone c=coneViaPoints(M);
insertCone(f,c);
fVector(f);
↦ 1,2,1
```

D.13.2.48 getCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: getCone(f,d,i[,m]); f fan, d int, i int, m int

Assume: d is between 0 and ambientDimension(f)
i is between 1 and numberOfConesOfDimension(f,d,o,m)

Return: cone, returns in the fan f of all cones in dimension d the i-th cone if m!=0, it will enumerate over maximal cones only

Example:

```
LIB "gfan.lib";
intmat M[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c=coneViaPoints(M);
fan f=emptyFan(3);
insertCone(f,c);
getCone(f,2,1,0);
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,0,1,
↦ 0,1,0
↦ LINEAR_SPAN
↦ -1,0,0
↦
getCone(f,2,2,0);
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,0,1,
↦ 1,0,0
↦ LINEAR_SPAN
↦ 0,-1,0
↦
```

D.13.2.49 insertCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: insertCone(f,c[,b]); f fan, c cone, b int

Assume: isCompatible(f,c)=1

Return: none, inserts the cone c into f
if b=0, then skips check whether f and c are compatible

Example:

```
LIB "gfan.lib";
fan f=emptyFan(3);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 3
↦
↦ DIM
↦ -1
↦
↦ LINEALITY_DIM
↦ 3
↦
↦ RAYS
↦
↦ N_RAYS
↦ 0
↦
↦ LINEALITY_SPACE
↦ 1 0 0 # 0
↦ 0 1 0 # 1
↦ 0 0 1 # 2
↦
↦ ORTH_LINEALITY_SPACE
↦
↦ F_VECTOR
↦
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦
↦ MAXIMAL_CONES
↦
intmat M[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c=coneViaPoints(M);
insertCone(f,c);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 3
↦
↦ DIM
↦ 3
↦
```

```
↦ LINEALITY_DIM
↦ 0
↦ 
↦ RAYS
↦ 0 0 1 # 0
↦ 0 1 0 # 1
↦ 1 0 0 # 2
↦ 
↦ N_RAYS
↦ 3
↦ 
↦ LINEALITY_SPACE
↦ 
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 # 0
↦ 0 -1 0 # 1
↦ 0 0 -1 # 2
↦ 
↦ F_VECTOR
↦ 1 3 3 1
↦ 
↦ SIMPLICIAL
↦ 1
↦ 
↦ PURE
↦ 1
↦ 
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 1} # Dimension 2
↦ {0 2}
↦ {1 2}
↦ {0 1 2} # Dimension 3
↦ 
↦ MAXIMAL_CONES
↦ {0 1 2} # Dimension 3
↦ 
```

D.13.2.50 isCompatible

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: isCompatible(f,c); f fan, c cone

Return: 1 if f and c live in the same ambient space and
if the intersection of c with any cone of f is a face of each; 0 otherwise

Example:

```
LIB "gfan.lib";
fan f=emptyFan(3);
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
isCompatible(f,c1);
↦ 1
insertCone(f,c1);
intmat M2[3][3]=
1,1,1,
1,0,0,
0,1,0;
cone c2=coneViaPoints(M2);
isCompatible(f,c2);
↦ 0
intmat M3[3][3]=
1,0,0,
0,1,0,
0,0,-1;
cone c3=coneViaPoints(M3);
isCompatible(f,c3);
↦ 1
```

D.13.2.51 isPure

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: isPure(f); f fan

Return: 1 if all maximal cones of f are of the same dimension 0 otherwise

Example:

```
LIB "gfan.lib";
fan f=fullFan(2);
isPure(f);
↦ 1
fan g=emptyFan(2);
intmat M1[2][2]=
1,0,
0,1;
cone c1=coneViaPoints(M1);
insertCone(g,c1);
isPure(g);
↦ 1
intmat M2[1][2]=
0,-1;
cone c2=coneViaPoints(M2);
insertCone(g,c2);
isPure(g,c2);
↦ 0
```

D.13.2.52 nmaxcones

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: nmaxcones(f); f fan

Return: int, the number of maximal cones in f

Example:

```
LIB "gfan.lib";
fan f=emptyFan(3);
nmaxcones(f);
↦ 0
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
insertCone(f,c1);
nmaxcones(f);
↦ 1
intmat M2[2][3]=
1,0,0,
0,-1,0;
cone c2=coneViaPoints(M2);
insertCone(f,c2);
nmaxcones(f);
↦ 2
```

D.13.2.53 ncones

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: ncones(f); f fan

Return: int, the number of cones in f

Example:

```
LIB "gfan.lib";
fan f=emptyFan(3);
ncones(f);
↦ 0
intmat M1[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c1=coneViaPoints(M1);
insertCone(f,c1);
ncones(f);
↦ 8
intmat M2[2][3]=
1,0,0,
0,-1,0;
cone c2=coneViaPoints(M2);
insertCone(f,c2);
ncones(f);
↦ 10
```

D.13.2.54 numberOfConesOfDimension

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: numberOfConesOfDimension(f,d[,m]); f fan, d int, m int

Assume: d is between 0 and ambientDimension(f)

Return: cone, returns in the fan f the number of cones in dimension d if m!=0, it will only count maximal cones

Example:

```
LIB "gfan.lib";
fan f=emptyFan(3);
ncones(f);
↦ 0
intmat M[3][3]=
1,0,0,
0,1,0,
0,0,1;
cone c=coneViaPoints(M);
insertCone(f,c);
numberOfConesOfDimension(f,0,0);
↦ 1
numberOfConesOfDimension(f,0,1);
↦ 0
numberOfConesOfDimension(f,1,0);
↦ 3
numberOfConesOfDimension(f,0,1);
↦ 0
numberOfConesOfDimension(f,2,0);
↦ 3
numberOfConesOfDimension(f,2,1);
↦ 0
numberOfConesOfDimension(f,3,0);
↦ 1
numberOfConesOfDimension(f,3,1);
↦ 1
```

D.13.2.55 removeCone

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: removeCone(f,c[,b]); f fan, c cone, b int

Assume: containsInCollection(f,c)=1

Return: none, removes the cone c from f
if b=0, skips the check whether c is contained in f

Example:

```
LIB "gfan.lib";
intmat M[2][2]=1,0,0,1;
intmat N[2][2]=1,0,1,-1;
cone c=coneViaPoints(M);
cone d=coneViaPoints(N);
fan f=emptyFan(2);
insertCone(f,c);
insertCone(f,d);
f;
```

```
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 0 1 # 0
↦ 1 -1 # 1
↦ 1 0 # 2
↦
↦ N_RAYS
↦ 3
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 3 2
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {2}
↦ {0 2} # Dimension 2
↦ {1 2}
↦
↦ MAXIMAL_CONES
↦ {0 2} # Dimension 2
↦ {1 2}
↦
removeCone(f,c);
f;
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
```

```
↦
↦ AMBIENT_DIM
↦ 2
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 0
↦
↦ RAYS
↦ 1 -1 # 0
↦ 1 0 # 1
↦
↦ N_RAYS
↦ 2
↦
↦ LINEALITY_SPACE
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 # 0
↦ 0 -1 # 1
↦
↦ F_VECTOR
↦ 1 2 1
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 0
↦ {0} # Dimension 1
↦ {1}
↦ {0 1} # Dimension 2
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 2
↦
```

D.13.2.56 dualPolytope

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: dualPolytope(p); p polytope

Return: polytope, the dual of p

Example:

```
LIB "gfan.lib";
intmat M[4][2]=
0,0,
1,0,
0,1,
1,1;
polytope p=polytopeViaPoints(M);
dualPolytope(p);
↦ AMBIENT_DIM
↦ 1
↦ INEQUALITIES
↦ 0,1,
↦ 1,0
↦ EQUATIONS
↦
↦
```

D.13.2.57 newtonPolytope

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: newtonPolytope(f); f poly

Return: polytope, the convex hull of all exponent vectors of f

Example:

```
LIB "gfan.lib";
ring r;
poly f=x+y+z;
polytope p=newtonPolytope(f);
p;
↦ AMBIENT_DIM
↦ 3
↦ INEQUALITIES
↦ 0,0,0,1,
↦ 0,0,1,0,
↦ 0,1,0,0
↦ EQUATIONS
↦ 1,-1,-1,-1
↦
```

D.13.2.58 vertices

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: vertices(p); p polytope

Return: bigintmat, the vertices of p modulo its lineality space

Example:

```
LIB "gfan.lib";
intmat M[4][3]=
1,0,0,
1,2,0,
1,0,2,
1,2,2,
1,1,1;
polytope p=polytopeViaPoints(M);
vertices(p);
```

```
↦ 1,2,0,
↦ 1,0,0,
↦ 1,0,2,
↦ 1,2,2
```

D.13.2.59 onesVector

Procedure from library `gfan.lib` (see Section D.13.2 [gfan_lib], page 2049).

Usage: onesVector(n); n int

Return: intvec, intvec of length n with all entries 1

Example:

```
LIB "gfan.lib";
intvec w = onesVector(3);
w;
↦ 1,1,1
```

D.13.3 gitfan_lib

Library: gitfan.lib

Purpose: Compute GIT-fans.

Authors: Janko Boehm, boehm at mathematik.uni-kl.de
Simon Keicher, keicher at mail.mathematik.uni-tuebingen.de
Yue Ren, ren at mathematik.uni-kl.de

Overview: This library allows you to calculate GIT-fans, torus orbits and GKZ-fans.

In provides features to make use of symmetries of the torus action under consideration.

The main procedure is GITfan which can be directly applied to an ideal and a grading matrix encoding the torus action, and returns a fan, the associated GIT-fan. We also provide various procedures implementing substeps of the algorithm to deal with large computations.

The library uses the package 'gfanlib' by Anders N. Jensen.

For notation, background, and algorithms see [BKR16].

Functions produce debug output if printlevel is positive.

Elements of the symmetric group Sn of type permutation can be created by the function permutationFromIntvec.
The images of 1,...,n can be obtained by permutationToIntvec. Composition of permutations can be done by the *-Operator, also powers can be computed in the usual way.

References:

[BKR16] J. Boehm, S. Keicher, Y. Ren: Computing GIT-Fans with Symmetry and the Mori Chamber Decomposition of M06bar, https://arxiv.org/abs/1603.09241

Types: permutation; Permutation in map representation.

Procedures:

D.13.3.1 isAface

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: isAface(a,gam0); a: ideal, gam0:intvec

Purpose: Checks whether gam0 is an a-face w.r.t. the ideal a.

Return: int

Example:

```
LIB "gitfan.lib";
ring R = 0,(T(1..4)),dp;
ideal I = T(1)*T(2)-T(4);
intvec w = 1,4;
intvec v = 1,2,4;
isAface(I,w); // should be 0
↦ 0
isAface(I,v); // should be 1
↦ 1
```

D.13.3.2 afaces

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: afaces(a [,L]); a: ideal, L: list of intvecs

Purpose: Returns a list of all a-faces (considered as intvecs of 0 and 1, where the i-th entry is 1 if the cone has the i-th unit basis vector as a generator), if L is specified only the faces of the simplex listed in L are considered (e.g. representatives with respect to a group action).

Return: a list of intvecs

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..3),dp;
ideal a = T(1)+T(2)+T(3);
list F = afaces(a);
print(F);
↦ [1]:
↦    0
↦ [2]:
↦    1,2
↦ [3]:
↦    1,3
↦ [4]:
↦    2,3
↦ [5]:
↦    1,2,3
print(size(F));
↦ 5
// 2nd ex //
ring R2 = 0,T(1..3),dp;
ideal a2 = T(2)^2*T(3)^2+T(1)*T(3);
list F2 = afaces(a2);
print(F2);
```

```
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    2
↦ [4]:
↦    1,2
↦ [5]:
↦    3
↦ [6]:
↦    1,2,3
print(size(F2));
↦ 6
// 3rd ex //
ring R3 = 0,T(1..3),dp;
ideal a3 = 0;
list F3 = afaces(a3);
print(F3);
↦ [1]:
↦    0
↦ [2]:
↦    1
↦ [3]:
↦    2
↦ [4]:
↦    1,2
↦ [5]:
↦    3
↦ [6]:
↦    1,3
↦ [7]:
↦    2,3
↦ [8]:
↦    1,2,3
print(size(F3));
↦ 8
// 4th ex //
ring R4 = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
list F4 = afaces(J);
print(size(F4));
↦ 172
```

D.13.3.3 fullDimImages

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: fullDimImages(afaces, Q); afaces: list, Q: intmat

Purpose: Determines the a-faces (represented as intvecs) from the list afaces which have a full-dimensional projection with respect to Q.

Return: a list of intvecs

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
size(AF);
↦ 81
size(fullDimImages(AF,Q));
↦ 36
```

D.13.3.4 minimalAfaces

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: minimalAfaces(listOfAfaces); listOfAfaces: list

Purpose: Returns a list of all minimal a-faces. Note that listOfAfaces must only contain afaces which project to full dimension.

Return: a list of intvecs

Example:

```
LIB "gitfan.lib";
setcores(4);
↦ 4
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
```

```
size(AF);
↦ 81
AF=fullDimImages(AF,Q);
size(AF);
↦ 36
AF=minimalAfaces(AF);
size(AF);
↦ 25
```

D.13.3.5 orbitCones

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: orbitCones(AF, Q[, d]); AF: list of intvecs, Q: intmat, d: int

Purpose: Returns the list consisting of all cones Q(gam0) where gam0 in AF. If the optional argument d is given then the function returns only the orbit cones of dimension at least d

Return: a list of cones

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J);
print(size(AF));
↦ 172
list OC = orbitCones(AF,Q);
size(OC);
↦ 172
```

D.13.3.6 GITcone

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITcone(OCcones, w); OCcones: list of orbit cones, w: bigintmat with one row

Purpose: Returns the intersection of all orbit cones containing w.

Return: cone,intvec with the GIT cone containing w, and the hash of this cone (the indices of the orbit cones contributing to the intersection)

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
AF=fullDimImages(AF,Q);
AF = minimalAfaces(AF);
list OC = orbitCones(AF,Q);
bigintmat w[1][nrows(Q)];
int j;
for(int i = 1; i <= nrows(Q); i++)
{
for(j=1;j<=nrows(Q);j++)
{
w[1,j]= w[1,j] + Q[1,i];
}
}
GITcone(OC,w);
↦ AMBIENT_DIM
↦ 5
↦ FACETS
↦ 0,0,0,0,1,
↦ 0,1,0,0,0
↦ LINEAR_SPAN
↦ 1,0,0,0,-1,
↦ 0,0,1,0,-1,
↦ 0,0,0,1,-1
↦  3,5,6,8,10,11,12,14,16,17,18,19,20,21,22,23,24,25
```

D.13.3.7 GITfan

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITfan(J,Q [, G]); J:ideal, Q:intmat, G:list

Purpose: Computes the GIT fan associated to J and Q. Optionally a symmetry group action on the column space of Q can be specified.

Return: a fan, the GIT fan.

Note: The procedure uses parallel computation for the construction of the GIT-cones. The a-faces are not computed in parallel. This can be done by calling the aface procedure specifying a list of simplex faces. If used with the optional argument G, the orbit decomposition of the simplex of columns of Q is computed. Refer to the Singular documentation on how to do this more efficiently using GAP.

Example:

```
LIB "gitfan.lib";
setcores(4);
↦ 4
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
fan GIT = GITfan(J,Q);
intmat Q[5][10] =
↦ // ** redefining Q (intmat Q[5][10] =) ./examples/GITfan.sing:17
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
fan GIT2 = GITfan(J,Q,simplexSymmetryGroup);
```

D.13.3.8 GITfanFromOrbitCones

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITfanFromOrbitCones(OC, Q, Qgamma [, file1, file2]); OC:list, Q:bigintmat, Qgamma:cone, file1:string, file2:string

Purpose: Returns the common refinement of the cones given in
the list OC which is supposed to contain the orbit cones intersected with Qgamma. The optional argument can be used to specify one or two strings with file names, where the first file will contain the hashes of the GIT-cones and the second argument the actual cones in their H-representation. To obtain the whole GIT-fan Qgamma has to be take the cone generated by the columns of Q.

Return: a list containing the bigint hashes of the GIT cones.

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
AF=fullDimImages(AF,Q);
AF = minimalAfaces(AF);
list OC = orbitCones(AF,Q);
cone Qgamma = coneViaPoints(transpose(Q));
list GIT = GITfanFromOrbitCones(OC,Q,Qgamma);
size(GIT);
↦ 76
```

D.13.3.9 GITfanParallel

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITfanParallel(OC, Q, Qgamma [, file1]); OC:list, Q:intmat, Qgamma:cone, file1:string

Purpose: Returns the common refinement of the cones given in
the list OC which is supposed to contain the orbit cones intersected with Qgamma. The optional argument can be used to specify a name for a file which will contain the hashes of the GIT-cones. To obtain the whole GIT-fan Qgamma has to be take the cone generated by the columns of Q.

Return: a list containing the bigint hashes of the GIT cones.

Note: The procedure uses parallel computation for the construction of the GIT-cones.

Example:

```
LIB "gitfan.lib";
setcores(4);
↦ 4
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J);
print(size(AF));
↦ 172
list OC = orbitCones(AF,Q);
cone Qgamma = coneViaPoints(transpose(Q));
list GIT = GITfanParallel(OC,Q,Qgamma);
size(GIT);
↦ 76
```

D.13.3.10 GKZfan

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

D.13.3.11 movingCone

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: movingCone(Q); Q: intmat

Purpose: Computes the moving cone from the grading matrix, with the degrees in the columns of Q.

Return: a cone

Example:

```
LIB "gitfan.lib";
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
cone mov = movingCone(Q);
mov;
↦ AMBIENT_DIM
↦ 5
↦ FACETS
↦ -2,1, 1, 1, 1,
↦ -1,1, 0, 0, 1,
↦ -1,1, 0, 1, 0,
↦ -1,1, 1, 0, 0,
↦  0,0, 0, 0, 1,
↦  0,0, 0, 1, 0,
↦  0,0, 1, 0, 0,
↦  1,0,-1, 0, 0,
↦  1,0, 0,-1, 0,
↦  1,0, 0, 0,-1
↦ LINEAR_SPAN
↦
↦
rays(mov);
↦ 1,1,0,1,0,
↦ 1,1,1,1,0,
↦ 1,2,0,0,0,
↦ 2,1,1,1,1,
↦ 1,1,1,0,0,
↦ 1,1,1,0,1,
↦ 1,1,0,1,1,
↦ 1,1,0,0,1,
↦ 1,0,1,1,1,
↦ 0,1,0,0,0
// moving cone where the image of the positive orthant
// is the entire space
// (from bug reported by Donton-Bury and Grab)
```

```
intmat Q2[3][16] =
7, 7,7,7, 7,7,7, 7,7,7, 7,7,7, -7,0,0,
0, 0,2,1, 0,2,1, 3,2,1, 3,2,4, 0,-3,0,
0, 0,1,2, 0,1,2, 3,1,2, 3,4,2, 0,0,-3;
cone mov2 = movingCone(Q2);
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
↦ // ** redefining Qdash (      intmat Qdash[nrows(Qt)-1][ncols(Qt)];) gitf\
  an.lib::movingCone:1012
mov2;
↦ AMBIENT_DIM
↦ 3
↦ FACETS
↦ 0,-1, 2,
↦ 0, 2,-1,
↦ 6,-7,-7
↦ LINEAR_SPAN
↦
↦
rays(mov2);
↦ 1,0,0,
↦ 7,2,4,
↦ 7,4,2
```

D.13.3.12 computeAfaceOrbits

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: computeAfaceOrbits(AF,G); AF list of intvecs, G: list of permutations

Purpose: Computes the orbits of the afaces in the list AF under the group action in G, where G is a list of permutations. We assume that the elements of G form a group and the first entry corresponds to the neutral element.

Return: a list of lists of intvecs

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
```

D.13.3.13 minimalAfaceOrbits

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: minimalAfaceOrbits(afaceOrbits); afaceOrbits: list

Purpose: Returns a list of all minimal a-face orbits.

Return: a list of intvecs

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 10 15
```

D.13.3.14 orbitConeOrbits

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: orbitConeOrbits(F, Q); F: list, Q: intmat

Purpose: Projects a list F of a-face orbits to the orbit cones with respect to Q. The function checks whether the projections are of full dimension and returns an error otherwise.

Return: a list of lists of cones

Example:

```
LIB "gitfan.lib";
// Note that simplexOrbitRepresentatives and simplexSymmetryGroup are subsets of the
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list simplexSymmetryGroup = permutationFromIntvec(intvec( 1 .. 10 )),
permutationFromIntvec(intvec( 1, 2, 4, 3, 5, 7, 6, 9, 8, 10 )),
permutationFromIntvec(intvec( 1, 3, 2, 4, 6, 5, 7, 8, 10, 9 )),
permutationFromIntvec(intvec( 1, 3, 4, 2, 6, 7, 5, 10, 8, 9 )),
permutationFromIntvec(intvec( 1, 4, 2, 3, 7, 5, 6, 9, 10, 8 )),
permutationFromIntvec(intvec( 1, 4, 3, 2, 7, 6, 5, 10, 9, 8 ));
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetry
apply(afaceOrbits,size);
↦ 3 3 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 3
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
```

D.13.3.15 minimalOrbitConeOrbits

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: minimalOrbitConeOrbits(listOfOrbits); listOfOrbits: list of lists of cones

Purpose: Minimizes a list of orbit cone orbits.

Return: a list of lists of cones

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 10 15
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
apply(listOfOrbitConeOrbits,size);
↦ 10 15
list listOfMinimalOrbitConeOrbits = minimalOrbitConeOrbits(listOfOrbitConeOrbits);
size(listOfMinimalOrbitConeOrbits);
↦ 2
```

D.13.3.16 intersectOrbitsWithMovingCone

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: intersectOrbitsWithMovingCone(OCmin, mov); OCmin: list of lists of cones, mov: cone

Purpose: Intersects all cones in the orbits in OCmin with mov discarting all orbits of cones which are not of full dimension. The resulting orbits are duplicate free.

Return: a list of lists of cones

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 10 15
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
cone mov = movingCone(Q);
```

```
intersectOrbitsWithMovingCone(listOfOrbitConeOrbits,mov);
↦ [1]:
↦    [1]:
↦       AMBIENT_DIM
↦ 5
↦ FACETS
↦ -2,1, 1, 1, 1,
↦ -1,1, 0, 0, 1,
↦ -1,1, 0, 1, 0,
↦ -1,1, 1, 0, 0,
↦  0,0, 0, 0, 1,
↦  0,0, 0, 1, 0,
↦  0,0, 1, 0, 0,
↦  1,0,-1, 0, 0,
↦  1,0, 0,-1, 0,
↦  1,0, 0, 0,-1
↦ LINEAR_SPAN
↦
↦
```

D.13.3.17 groupActionOnQImage

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: groupActionOnQImage(G,Q); G: list of permutations, Q: intmat

Purpose: Given the group G of permutations acting on the simplex on ncols(Q) objects, computes the corresponding group action on the image of Q. We assume that the basering has characteristic 0.

Return: list of matrices

Example:

```
LIB "gitfan.lib";
ring R = 0,(x),dp;
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list generatorsG = permutationFromIntvec(intvec( 1, 3, 2, 4, 6, 5, 7, 8, 10, 9 )),
permutationFromIntvec(intvec( 5, 7, 1, 6, 9, 2, 8, 4, 10, 3 ));
groupActionOnQImage(generatorsG,Q);
↦ [1]:
↦    1,0,0,0,0,
↦    0,1,0,0,0,
↦    0,0,1,0,0,
↦    0,0,0,0,1,
↦    0,0,0,1,0
↦ [2]:
↦    -1,1,1,0,1,
↦    1,0,0,0,0,
↦    -1,1,0,0,1,
↦    -1,1,1,0,0,
↦    -2,1,1,1,1
```

D.13.3.18 groupActionOnHashes

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: groupActionOnHashes(Asigma,OCmov); Asigma: list, OCmov: list of list of cones

Purpose: From the list of orbits of orbitcones, and the symmetry group representation given by the matrices in Asigma, compute the corresponding permutation representation of the symmetry group on the orbit cones. The permutations are specified in a map representation of length the sum of the size of the orbits of OCmov.

Return: list of permutations

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J);
list OC = orbitCones(AF,Q);
list generatorsG = permutationFromIntvec(intvec( 1, 3, 2, 4, 6, 5, 7, 8, 10, 9 )),
permutationFromIntvec(intvec( 5, 7, 1, 6, 9, 2, 8, 4, 10, 3 ));
list Asigmagens = groupActionOnQImage(generatorsG,Q);
groupActionOnHashes(Asigmagens,list(OC));
↦ [1]:
↦    |   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17 \
   18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  \
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  5\
  4  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72\
    73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 \
   91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 1\
  09 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 12\
  7 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145\
   146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 \
  164 165 166 167 168 169 170 171 172|
↦ |   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0\
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 \
    0   0   0   0   0   0   0  44   0   0   0   0   0   0   0   0   0   0  \
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   \
  0   0   0   0   0   0   0   0   0   0   0   0   0 120   0   0   0 124   0\
   126 127 128 129   0   0   0   0   0   0   0   0   0   0   0   0   0   0 \
    0   0   0   0   0   0   0   0   0   0  86   0   0   0  90   0  92  93  \
  94  95   0   0   0   0   0   0   0 137   0   0   0 141   0 143 144 145 14\
   6   0   0   0   0   0   0 153 155 154 156   0 163 165 164 166   0 158 160\
    159 161   0 168 170 169 171 172|
↦
↦  [2]:
↦     |   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17 \
    18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  \
   36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  5\
   4  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72\
     73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90 \
    91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 1\
   09 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 12\
   7 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145\
    146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 \
   164 165 166 167 168 169 170 171 172|
↦  |   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0\
      0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 \
     0   0   0   0   0   0   0 120   0   0   0   0   0   0   0   0   0   0  \
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   \
   0   0   0   0   0   0   0   0   0   0   0   0   0 126   0   0   0 128   0\
     44 124 127 129   0   0   0   0   0   0   0   0   0   0   0   0   0   0 \
     0   0   0   0   0   0   0   0   0   0 165   0   0   0 163   0 141  90 1\
   64 166   0   0   0   0   0   0   0 155   0   0   0 153   0 137  86 154 15\
   6   0   0   0   0   0   0 144  93 160 170   0 143  92 158 168   0 145  94\
    159 171   0 146  95 161 169 172|
↦
list simplexSymmetryGroup = G25Action();
list orb = findOrbits(simplexSymmetryGroup,nrows(Q));
list simplexOrbitRepresentatives;
for (int i=1;i<=size(orb);i++){simplexOrbitRepresentatives[i]=orb[i][1];}
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
list listOfMinimalOrbitConeOrbits = minimalOrbitConeOrbits(listOfOrbitConeOrbits);
list Asigma = groupActionOnQImage(simplexSymmetryGroup,Q);
groupActionOnHashes(Asigma,listOfOrbitConeOrbits);
↦  [1]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦  |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24\
    25|
↦
↦  [2]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦  |  1  2  4  3  5  7  6  9  8 10 11 13 12 14 16 15 17 19 18 23 24 25 20 21\
    22|
↦
↦  [3]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦  |  1  3  2  4  6  5  7  8 10  9 12 11 13 15 14 16 20 21 22 17 18 19 23 25\
   24|
↦
↦ [4]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  4  2  3  7  5  6  9 10  8 13 11 12 16 14 15 23 24 25 17 19 18 20 22\
   21|
↦
↦ [5]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  3  4  2  6  7  5 10  8  9 12 13 11 15 16 14 20 22 21 23 25 24 17 18\
   19|
↦
↦ [6]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  4  3  2  7  6  5 10  9  8 13 12 11 16 15 14 23 25 24 20 22 21 17 19\
   18|
↦
↦ [7]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  5  6  7  2  3  4  8  9 10 14 15 16 11 12 13 17 19 18 20 22 21 23 25\
   24|
↦
↦ [8]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  5  7  6  2  4  3  9  8 10 14 16 15 11 13 12 17 18 19 23 25 24 20 22\
   21|
↦
↦ [9]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  6  5  7  3  2  4  8 10  9 15 14 16 12 11 13 20 22 21 17 19 18 23 24\
   25|
↦
↦ [10]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  7  5  6  4  2  3  9 10  8 16 14 15 13 11 12 23 25 24 17 18 19 20 21\
   22|
↦
↦ [11]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  6  7  5  3  4  2 10  8  9 15 16 14 12 13 11 20 21 22 23 24 25 17 19\
   18|
↦
↦ [12]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  1  7  6  5  4  3  2 10  9  8 16 15 14 13 12 11 23 24 25 20 21 22 17 18\
   19|
↦ 
↦ [13]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  1  3  4  5  8  9  6  7 10 11 13 12 17 18 19 14 15 16 21 20 22 24 23\
   25|
↦ 
↦ [14]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  1  4  3  5  9  8  7  6 10 11 12 13 17 19 18 14 16 15 24 23 25 21 20\
   22|
↦ 
↦ [15]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  1  2  4  6  8 10  5  7  9 12 13 11 20 21 22 15 14 16 18 17 19 25 23\
   24|
↦ 
↦ [16]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  1  2  3  7  9 10  5  6  8 13 12 11 23 24 25 16 14 15 19 17 18 22 20\
   21|
↦ 
↦ [17]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  1  4  2  6 10  8  7  5  9 12 11 13 20 22 21 15 16 14 25 23 24 18 17\
   19|
↦ 
↦ [18]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  1  3  2  7 10  9  6  5  8 13 11 12 23 25 24 16 15 14 22 20 21 19 17\
   18|
↦ 
↦ [19]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  1  6  7  2  8  9  3  4 10 14 16 15 17 19 18 11 12 13 22 20 21 25 23\
   24|
↦ 
↦ [20]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  1  7  6  2  9  8  4  3 10 14 15 16 17 18 19 11 13 12 25 23 24 22 20\
   21|
↦ 
↦ [21]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  1  5  7  3  8 10  2  4  9 15 16 14 20 22 21 12 11 13 19 17 18 24 23\
   25|
↦
↦ [22]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  1  5  6  4  9 10  2  3  8 16 15 14 23 25 24 13 11 12 18 17 19 21 20\
   22|
↦
↦ [23]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  1  7  5  3 10  8  4  2  9 15 14 16 20 21 22 12 13 11 24 23 25 19 17\
   18|
↦
↦ [24]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  1  6  5  4 10  9  3  2  8 16 14 15 23 24 25 13 12 11 21 20 22 18 17\
   19|
↦
↦ [25]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  3  1  4  8  5  9  6 10  7 13 11 12 18 17 19 21 20 22 14 15 16 24 25\
   23|
↦
↦ [26]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  4  1  3  9  5  8  7 10  6 12 11 13 19 17 18 24 23 25 14 16 15 21 22\
   20|
↦
↦ [27]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  2  1  4  8  6 10  5  9  7 13 12 11 21 20 22 18 17 19 15 14 16 25 24\
   23|
↦
↦ [28]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  2  1  3  9  7 10  5  8  6 12 13 11 24 23 25 19 17 18 16 14 15 22 21\
   20|
↦
↦ [29]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  4  1  2 10  6  8  7  9  5 11 12 13 22 20 21 25 23 24 15 16 14 18 19\
   17|
↦
↦ [30]:
```

```
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  3  1  2 10  7  9  6  8  5 11 13 12 25 23 24 22 20 21 16 15 14 19 18\
   17|
↦
↦ [31]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  6  1  7  8  2  9  3 10  4 16 14 15 19 17 18 22 20 21 11 12 13 25 24\
   23|
↦
↦ [32]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  7  1  6  9  2  8  4 10  3 15 14 16 18 17 19 25 23 24 11 13 12 22 21\
   20|
↦
↦ [33]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  5  1  7  8  3 10  2  9  4 16 15 14 22 20 21 19 17 18 12 11 13 24 25\
   23|
↦
↦ [34]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  5  1  6  9  4 10  2  8  3 15 16 14 25 23 24 18 17 19 13 11 12 21 22\
   20|
↦
↦ [35]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  7  1  5 10  3  8  4  9  2 14 15 16 21 20 22 24 23 25 12 13 11 19 18\
   17|
↦
↦ [36]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  6  1  5 10  4  9  3  8  2 14 16 15 24 23 25 21 20 22 13 12 11 18 19\
   17|
↦
↦ [37]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  3  4  1  8  9  5 10  6  7 13 12 11 18 19 17 21 22 20 24 25 23 14 15\
   16|
↦
↦ [38]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  4  3  1  9  8  5 10  7  6 12 13 11 19 18 17 24 25 23 21 22 20 14 16\
   15|
↦
```

```
↦ [39]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  2  4  1  8 10  6  9  5  7 13 11 12 21 22 20 18 19 17 25 24 23 15 14\
   16|
↦ 
↦ [40]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  2  3  1  9 10  7  8  5  6 12 11 13 24 25 23 19 18 17 22 21 20 16 14\
   15|
↦ 
↦ [41]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  4  2  1 10  8  6  9  7  5 11 13 12 22 21 20 25 24 23 18 19 17 15 16\
   14|
↦ 
↦ [42]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  3  2  1 10  9  7  8  6  5 11 12 13 25 24 23 22 21 20 19 18 17 16 15\
   14|
↦ 
↦ [43]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  6  7  1  8  9  2 10  3  4 16 15 14 19 18 17 22 21 20 25 24 23 11 12\
   13|
↦ 
↦ [44]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  7  6  1  9  8  2 10  4  3 15 16 14 18 19 17 25 24 23 22 21 20 11 13\
   12|
↦ 
↦ [45]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  5  7  1  8 10  3  9  2  4 16 14 15 22 21 20 19 18 17 24 25 23 12 11\
   13|
↦ 
↦ [46]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  5  6  1  9 10  4  8  2  3 15 14 16 25 24 23 18 19 17 21 22 20 13 11\
   12|
↦ 
↦ [47]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  7  5  1 10  8  3  9  4  2 14 16 15 21 22 20 24 25 23 19 18 17 12 13\
   11|
↦
↦ [48]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  6  5  1 10  9  4  8  3  2 14 15 16 24 25 23 21 22 20 18 19 17 13 12\
   11|
↦
↦ [49]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  5  8  9  1  3  4  6  7 10 17 18 19 11 13 12 14 16 15 21 22 20 24 25\
   23|
↦
↦ [50]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  5  9  8  1  4  3  7  6 10 17 19 18 11 12 13 14 15 16 24 25 23 21 22\
   20|
↦
↦ [51]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  2  8  9  1  6  7  3  4 10 17 19 18 14 16 15 11 13 12 22 21 20 25 24\
   23|
↦
↦ [52]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  2  9  8  1  7  6  4  3 10 17 18 19 14 15 16 11 12 13 25 24 23 22 21\
   20|
↦
↦ [53]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  6  8 10  1  2  4  5  7  9 20 21 22 12 13 11 15 16 14 18 19 17 25 24\
   23|
↦
↦ [54]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  7  9 10  1  2  3  5  6  8 23 24 25 13 12 11 16 15 14 19 18 17 22 21\
   20|
↦
↦ [55]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  6 10  8  1  4  2  7  5  9 20 22 21 12 11 13 15 14 16 25 24 23 18 19\
   17|
↦
↦ [56]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  7 10  9  1  3  2  6  5  8 23 25 24 13 11 12 16 14 15 22 21 20 19 18\
   17|
↦ 
↦ [57]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  3  8 10  1  5  7  2  4  9 20 22 21 15 16 14 12 13 11 19 18 17 24 25\
   23|
↦ 
↦ [58]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  4  9 10  1  5  6  2  3  8 23 25 24 16 15 14 13 12 11 18 19 17 21 22\
   20|
↦ 
↦ [59]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  3 10  8  1  7  5  4  2  9 20 21 22 15 14 16 12 11 13 24 25 23 19 18\
   17|
↦ 
↦ [60]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  4 10  9  1  6  5  3  2  8 23 24 25 16 14 15 13 11 12 21 22 20 18 19\
   17|
↦ 
↦ [61]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  8  5  9  3  1  4  6 10  7 18 17 19 13 11 12 21 22 20 14 16 15 24 23\
   25|
↦ 
↦ [62]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  9  5  8  4  1  3  7 10  6 19 17 18 12 11 13 24 25 23 14 15 16 21 20\
   22|
↦ 
↦ [63]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  8  2  9  6  1  7  3 10  4 19 17 18 16 14 15 22 21 20 11 13 12 25 23\
   24|
↦ 
↦ [64]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  9  2  8  7  1  6  4 10  3 18 17 19 15 14 16 25 24 23 11 12 13 22 20\
   21|
↦ 
↦ [65]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  8  6 10  2  1  4  5  9  7 21 20 22 13 12 11 18 19 17 15 16 14 25 23\
    24|
↦
↦ [66]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  4  9  7 10  2  1  3  5  8  6 24 23 25 12 13 11 19 18 17 16 15 14 22 20\
    21|
↦
↦ [67]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  3 10  6  8  4  1  2  7  9  5 22 20 21 11 12 13 25 24 23 15 14 16 18 17\
    19|
↦
↦ [68]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  4 10  7  9  3  1  2  6  8  5 25 23 24 11 13 12 22 21 20 16 14 15 19 17\
    18|
↦
↦ [69]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  6  8  3 10  5  1  7  2  9  4 22 20 21 16 15 14 19 18 17 12 13 11 24 23\
    25|
↦
↦ [70]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  7  9  4 10  5  1  6  2  8  3 25 23 24 15 16 14 18 19 17 13 12 11 21 20\
    22|
↦
↦ [71]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  6 10  3  8  7  1  5  4  9  2 21 20 22 14 15 16 24 25 23 12 11 13 19 17\
    18|
↦
↦ [72]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  7 10  4  9  6  1  5  3  8  2 24 23 25 14 16 15 21 22 20 13 11 12 18 17\
    19|
↦
↦ [73]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  2  8  9  5  3  4  1 10  6  7 18 19 17 13 12 11 21 20 22 24 23 25 14 16\
    15|
↦
↦ [74]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  2  9  8  5  4  3  1 10  7  6 19 18 17 12 13 11 24 23 25 21 20 22 14 15\
   16|
↦
↦ [75]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  8  9  2  6  7  1 10  3  4 19 18 17 16 15 14 22 20 21 25 23 24 11 13\
   12|
↦
↦ [76]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  5  9  8  2  7  6  1 10  4  3 18 19 17 15 16 14 25 23 24 22 20 21 11 12\
   13|
↦
↦ [77]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3  8 10  6  2  4  1  9  5  7 21 22 20 13 11 12 18 17 19 25 23 24 15 16\
   14|
↦
↦ [78]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4  9 10  7  2  3  1  8  5  6 24 25 23 12 11 13 19 17 18 22 20 21 16 15\
   14|
↦
↦ [79]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  3 10  8  6  4  2  1  9  7  5 22 21 20 11 13 12 25 23 24 18 17 19 15 14\
   16|
↦
↦ [80]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  4 10  9  7  3  2  1  8  6  5 25 24 23 11 12 13 22 20 21 19 17 18 16 14\
   15|
↦
↦ [81]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6  8 10  3  5  7  1  9  2  4 22 21 20 16 14 15 19 17 18 24 23 25 12 13\
   11|
↦
↦ [82]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7  9 10  4  5  6  1  8  2  3 25 24 23 15 14 16 18 17 19 21 20 22 13 12\
   11|
↦
↦ [83]:
```

```
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  6 10  8  3  7  5  1  9  4  2 21 22 20 14 16 15 24 23 25 19 17 18 12 11\
   13|
↦
↦ [84]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  7 10  9  4  6  5  1  8  3  2 24 25 23 14 15 16 21 20 22 18 17 19 13 11\
   12|
↦
↦ [85]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  2  5  9  3  6 10  1  4  7 18 19 17 21 22 20 13 11 12 16 14 15 23 24\
   25|
↦
↦ [86]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  2  5  8  4  7 10  1  3  6 19 18 17 24 25 23 12 11 13 15 14 16 20 21\
   22|
↦
↦ [87]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  5  2  9  6  3 10  1  7  4 19 18 17 22 21 20 16 14 15 13 11 12 23 25\
   24|
↦
↦ [88]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  5  2  8  7  4 10  1  6  3 18 19 17 25 24 23 15 14 16 12 11 13 20 22\
   21|
↦
↦ [89]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  3  6 10  2  5  9  1  4  7 21 22 20 18 19 17 13 12 11 16 15 14 23 25\
   24|
↦
↦ [90]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  4  7 10  2  5  8  1  3  6 24 25 23 19 18 17 12 13 11 15 16 14 20 22\
   21|
↦
↦ [91]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  6  3 10  5  2  9  1  7  4 22 21 20 19 18 17 16 15 14 13 12 11 23 24\
   25|
↦
```

```
↦ [92]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  7  4 10  5  2  8  1  6  3 25 24 23 18 19 17 15 16 14 12 13 11 20 21\
   22|
↦ 
↦ [93]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  3  6  8  4  7  9  1  2  5 22 21 20 25 24 23 11 12 13 14 15 16 17 18\
   19|
↦ 
↦ [94]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  4  7  9  3  6  8  1  2  5 25 24 23 22 21 20 11 13 12 14 16 15 17 19\
   18|
↦ 
↦ [95]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  6  3  8  7  4  9  1  5  2 21 22 20 24 25 23 14 15 16 11 12 13 17 19\
   18|
↦ 
↦ [96]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  7  4  9  6  3  8  1  5  2 24 25 23 21 22 20 14 16 15 11 13 12 17 18\
   19|
↦ 
↦ [97]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  2  9  5  3 10  6  4  1  7 18 17 19 21 20 22 13 12 11 23 24 25 16 14\
   15|
↦ 
↦ [98]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  2  8  5  4 10  7  3  1  6 19 17 18 24 23 25 12 13 11 20 21 22 15 14\
   16|
↦ 
↦ [99]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  5  9  2  6 10  3  7  1  4 19 17 18 22 20 21 16 15 14 23 25 24 13 11\
   12|
↦ 
↦ [100]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  5  8  2  7 10  4  6  1  3 18 17 19 25 23 24 15 16 14 20 22 21 12 11\
   13|
↦
↦ [101]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  8  3 10  6  2  9  5  4  1  7 21 20 22 18 17 19 13 11 12 23 25 24 16 15\
    14|
↦
↦ [102]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  9  4 10  7  2  8  5  3  1  6 24 23 25 19 17 18 12 11 13 20 22 21 15 16\
    14|
↦
↦ [103]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  8  6 10  3  5  9  2  7  1  4 22 20 21 19 17 18 16 14 15 23 24 25 13 12\
    11|
↦
↦ [104]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  9  7 10  4  5  8  2  6  1  3 25 23 24 18 17 19 15 14 16 20 21 22 12 13\
    11|
↦
↦ [105]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ | 10  3  8  6  4  9  7  2  1  5 22 20 21 25 23 24 11 13 12 17 18 19 14 15\
    16|
↦
↦ [106]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ | 10  4  9  7  3  8  6  2  1  5 25 23 24 22 20 21 11 12 13 17 19 18 14 16\
    15|
↦
↦ [107]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ | 10  6  8  3  7  9  4  5  1  2 21 20 22 24 23 25 14 16 15 17 19 18 11 12\
    13|
↦
↦ [108]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ | 10  7  9  4  6  8  3  5  1  2 24 23 25 21 20 22 14 15 16 17 18 19 11 13\
    12|
↦
↦ [109]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
    24 25|
↦ |  8  9  2  5 10  3  6  4  7  1 17 18 19 20 21 22 23 24 25 13 12 11 16 15\
   14|
↦
↦ [110]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  8  2  5 10  4  7  3  6  1 17 19 18 23 24 25 20 21 22 12 13 11 15 16\
   14|
↦
↦ [111]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8  9  5  2 10  6  3  7  4  1 17 19 18 20 22 21 23 25 24 16 15 14 13 12\
   11|
↦
↦ [112]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9  8  5  2 10  7  4  6  3  1 17 18 19 23 25 24 20 22 21 15 16 14 12 13\
   11|
↦
↦ [113]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8 10  3  6  9  2  5  4  7  1 20 21 22 17 18 19 23 25 24 13 11 12 16 14\
   15|
↦
↦ [114]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9 10  4  7  8  2  5  3  6  1 23 24 25 17 19 18 20 22 21 12 11 13 15 14\
   16|
↦
↦ [115]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  8 10  6  3  9  5  2  7  4  1 20 22 21 17 19 18 23 24 25 16 14 15 13 11\
   12|
↦
↦ [116]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ |  9 10  7  4  8  5  2  6  3  1 23 25 24 17 18 19 20 21 22 15 14 16 12 11\
   13|
↦
↦ [117]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  8  3  6  9  4  7  2  5  1 20 22 21 23 25 24 17 18 19 11 13 12 14 16\
   15|
↦
↦ [118]:
↦     |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  9  4  7  8  3  6  2  5  1 23 25 24 20 22 21 17 19 18 11 12 13 14 15\
   16|
↦ 
↦ [119]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  8  6  3  9  7  4  5  2  1 20 21 22 23 24 25 17 19 18 14 16 15 11 13\
   12|
↦ 
↦ [120]:
↦    |  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23\
   24 25|
↦ | 10  9  7  4  8  6  3  5  2  1 23 24 25 20 21 22 17 18 19 14 15 16 11 12\
   13|
↦ 
```

D.13.3.19 storeActionOnOrbitConeIndices

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: storeActionOnOrbitConeIndices(generatorsGperm,st); generatorsGperm: list, fn: string

Purpose: Write the action on the set of orbit cones to the file fn in Singular readable format.

Return: nothing

D.13.3.20 permutationFromIntvec

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: permutationFromIntvec(sigmaImage); sigmaImage: intvec

Purpose: Create a permutation from an intvec of images.

Return: permutation

D.13.3.21 permutationToIntvec

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: permutationToIntvec(sigma); sigma: permutation

Purpose: Convert a permutation to an intvec of images.

Return: intvec

Example:

```
LIB "gitfan.lib";
permutation sigma = permutationFromIntvec(intvec( 1, 2, 4, 3, 5, 7, 6, 9, 8, 10 ));
sigma;
↦ |  1  2  3  4  5  6  7  8  9 10|
↦ |  1  2  4  3  5  7  6  9  8 10|
↦ 
permutationToIntvec(sigma);
↦ 1,2,4,3,5,7,6,9,8,10
```

D.13.3.22 evaluateProduct

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: evaluateProduct(generatorsGperm,st); generatorsGperm: list, st: string

Purpose: Evaluates a formal product of variables xi in st, where xi corresponds to the permutation generatorsGperm[i].

Return: permutation

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J);
list OC = orbitCones(AF,Q);
list generatorsG = permutationFromIntvec(intvec( 1, 3, 2, 4, 6, 5, 7, 8, 10, 9 )),
permutationFromIntvec(intvec( 5, 7, 1, 6, 9, 2, 8, 4, 10, 3 ));
list Asigmagens = groupActionOnQImage(generatorsG,Q);
//list actionOnOrbitconeIndicesForGenerators = groupActionOnHashes(Asigmagens,OC);
string elementInTermsOfGenerators =
"(x2^-1*x1^-1)^3*x1^-1";
//evaluateProduct(actionOnOrbitconeIndicesForGenerators, elementInTermsOfGenerators)
```

D.13.3.23 GITfanSymmetric

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITfanSymmetric(OC, Q, Qgamma, actiononorbitcones [, file1, file2]); OC:list, Q:bigintmat, Qgamma:cone, actiononorbitcones: list of intvec, file1:string, file2:string

Purpose: Returns the common refinement of the cones given in
the list OC which is supposed to contain the orbit cones intersected with Qgamma. The list actiononorbitcones is supposed to contain the symmetry group acting as permutations of on the list of orbit cones in OC. The optional argument can be used to specify one or two strings with file names, where the first file will contain the hashes of the GIT-cones and the second argument the actual cones in their H-representation. To obtain the whole GIT-fan Qgamma has to be take the cone generated by the columns of Q.

Return: a list containing the bigint hashes of the GIT cones.

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 10 15
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
apply(listOfOrbitConeOrbits,size);
↦ 10 15
list listOfMinimalOrbitConeOrbits = minimalOrbitConeOrbits(listOfOrbitConeOrbits);
size(listOfMinimalOrbitConeOrbits);
↦ 2
list Asigma = groupActionOnQImage(simplexSymmetryGroup,Q);
list actionOnOrbitconeIndices = groupActionOnHashes(Asigma,listOfOrbitConeOrbits);
list OClist = listOfOrbitConeOrbits[1];
for (int i =2;i<=size(listOfOrbitConeOrbits);i++){
```

```
OClist = OClist + listOfOrbitConeOrbits[i];
}
cone mov = coneViaPoints(transpose(Q));
mov = canonicalizeCone(mov);
printlevel = 3;
list Sigma = GITfanSymmetric(OClist, Q, mov, actionOnOrbitconeIndices);
↦ testing 215, 230, 113, 58, 85
↦ computed cone of dimension 5
↦ time for facets: 0
↦ overall: 2   open: 1   time for loop: 0
↦ time for facets: 0
↦ overall: 3   open: 1   time for loop: 0
↦ time for facets: 0
↦ overall: 5   open: 2   time for loop: 0
↦ time for facets: 0
↦ overall: 6   open: 2   time for loop: 0
↦ time for facets: 0
↦ overall: 6   open: 1   time for loop: 0
↦ time for facets: 0
↦ overall: 6   open: 0   time for loop: 0
Sigma;
↦ [1]:
↦    7183
↦ [2]:
↦    224275
↦ [3]:
↦    605191
↦ [4]:
↦    4946947
↦ [5]:
↦    14416897
↦ [6]:
↦    33553408
```

D.13.3.24 GITfanParallelSymmetric

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: GITfanParallelSymmetric(OC, Q, Qgamma, actiononorbitcones [, file1]); OC:list, Q:bigintmat, Qgamma:cone, actiononorbitcones: list of intvec, file1:string

Purpose: Returns the common refinement of the cones given in
the list OC which is supposed to contain the orbit cones intersected with Qgamma. The list actiononorbitcones is supposed to contain the symmetry group acting as permutations of on the list of orbit cones in OC. The optional argument can be used to specify a name for a file which will contain the hashes of the GIT-cones. To obtain the whole GIT-fan Qgamma has to be take the cone generated by the columns of Q.

Return: a list containing the bigint hashes of the GIT cones.

Note: The procedure uses parallel computation for the construction of the GIT-cones.

Example:

```
LIB "gitfan.lib";
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
```

```
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list simplexSymmetryGroup = G25Action();
list simplexOrbitRepresentatives = intvec( 1, 2, 3, 4, 5 ),
intvec( 1, 2, 3, 5, 6 ),
intvec( 1, 2, 3, 5, 7 ),
intvec( 1, 2, 3, 5, 10 ),
intvec( 1, 2, 3, 7, 9 ),
intvec( 1, 2, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6 ),
intvec( 1, 2, 3, 4, 5, 10 ),
intvec( 1, 2, 3, 5, 6, 8 ),
intvec( 1, 2, 3, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 7, 10 ),
intvec( 1, 2, 3, 7, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7 ),
intvec( 1, 2, 3, 4, 5, 6, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9 ),
intvec( 1, 2, 3, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8 ),
intvec( 1, 2, 3, 4, 5, 6, 9, 10 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9 ),
intvec( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );
list afaceOrbitRepresentatives=afaces(J,simplexOrbitRepresentatives);
list fulldimAfaceOrbitRepresentatives=fullDimImages(afaceOrbitRepresentatives,Q);
list afaceOrbits=computeAfaceOrbits(fulldimAfaceOrbitRepresentatives,simplexSymmetryG
apply(afaceOrbits,size);
↦ 10 15 10 1
list minAfaceOrbits = minimalAfaceOrbits(afaceOrbits);
apply(minAfaceOrbits,size);
↦ 10 15
list listOfOrbitConeOrbits = orbitConeOrbits(minAfaceOrbits,Q);
apply(listOfOrbitConeOrbits,size);
↦ 10 15
list listOfMinimalOrbitConeOrbits = minimalOrbitConeOrbits(listOfOrbitConeOrbits);
size(listOfMinimalOrbitConeOrbits);
↦ 2
list Asigma = groupActionOnQImage(simplexSymmetryGroup,Q);
list actionOnOrbitconeIndices = groupActionOnHashes(Asigma,listOfOrbitConeOrbits);
list OClist = listOfOrbitConeOrbits[1];
for (int i =2;i<=size(listOfOrbitConeOrbits);i++){
OClist = OClist + listOfOrbitConeOrbits[i];
}
```

```
cone mov = coneViaPoints(transpose(Q));
mov = canonicalizeCone(mov);
list Sigma = GITfanParallelSymmetric(OClist, Q, mov, actionOnOrbitconeIndices);
Sigma;
↦ [1]:
↦    7183
↦ [2]:
↦    224275
↦ [3]:
↦    605191
↦ [4]:
↦    4946947
↦ [5]:
↦    14416897
↦ [6]:
↦    33553408
```

D.13.3.25 bigintToBinary

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: bigintToBinary(n, r): n bigint, r int.

Assume: n is smaller then 2^r.

Return: an intvec, with entries the positions of 1 in the binary representation of n with r bits.

Example:

```
LIB "gitfan.lib";
bigintToBinary(bigint(2)^90-1, 90);
↦ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,2\
   8,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52\
   ,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,\
   77,78,79,80,81,82,83,84,85,86,87,88,89,90
```

D.13.3.26 binaryToBigint

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

D.13.3.27 applyPermutationToIntvec

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: applyPermutationToIntvec(v,g); v: intvec, g:permutation

Purpose: Apply g to the set of entries of v. The result is a sorted intvec. We assume that the entries of v are valid arguments of g. We do not require that the input is sorted.

Return: intvec.

Example:

```
LIB "gitfan.lib";
permutation g = permutationFromIntvec(intvec(10, 9, 7, 4, 8, 6, 3, 5, 2, 1));
applyPermutationToIntvec(intvec(1, 3, 4, 6, 8),g);
↦ 4,5,6,7,10
```

D.13.3.28 hashToCone

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: hashToCone(v, OC): v bigint, OC list of cones.

Assume: the elements of OC are the orbit cones used in the hash representation of the GIT cones.

Return: a cone, the intersection of the cones in OC according to the binary representation of the hash v.

Example:

```
LIB "gitfan.lib";
setcores(4);
↦ 4
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
AF=fullDimImages(AF,Q);
AF = minimalAfaces(AF);
list OC = orbitCones(AF,Q);
bigint v = 21300544;
hashToCone(v, OC);
↦ AMBIENT_DIM
↦ 5
↦ FACETS
↦ -1, 0,0, 0,1,
↦ -1, 0,1, 0,0,
↦  0, 0,0,-1,0,
↦  0, 1,0, 1,0,
↦  1,-1,0,-1,0
↦ LINEAR_SPAN
↦ 
↦ 
```

D.13.3.29 hashesToFan

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: hashesToFan(hashes, OC): hashes list of bigint, OC list of cones.

Assume: the elements of OC are the orbit cones used in the hash representation of the GIT cones.

Return: a fan, with maximal cones the intersections of the cones in OC according to the binary representation of the hashes.

Example:

```
LIB "gitfan.lib";
setcores(4);
↦ 4
ring R = 0,T(1..10),wp(1,1,1,1,1,1,1,1,1,1);
ideal J =
T(5)*T(10)-T(6)*T(9)+T(7)*T(8),
T(1)*T(9)-T(2)*T(7)+T(4)*T(5),
T(1)*T(8)-T(2)*T(6)+T(3)*T(5),
T(1)*T(10)-T(3)*T(7)+T(4)*T(6),
T(2)*T(10)-T(3)*T(9)+T(4)*T(8);
intmat Q[5][10] =
1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1, 1, 0, 0, 0, -1, 1, 0, 0,
0, 1, 0, 1, 0, -1, 0, 0, 1, 0,
0, 0, 1, 1, -1, 0, 0, 0, 0, 1;
list AF= afaces(J,nrows(Q));
AF=fullDimImages(AF,Q);
AF = minimalAfaces(AF);
list OC = orbitCones(AF,Q);
cone Qgamma = coneViaPoints(transpose(Q));
list GIT = GITfanParallel(OC,Q,Qgamma);
fan Sigma = hashesToFan(GIT,OC);
```

D.13.3.30 gitCone

Procedure from library `gitfan.lib` (see Section D.13.3 [gitfan_lib], page 2089).

Usage: gitCone(a, Q, w); a: ideal, Q:bigintmat, w:bigintmat

Purpose: Returns the GIT-cone lambda(w), i.e. the intersection of all orbit cones containing the vector w.

Note: call this only if you are interested in a single GIT-cone.

Return: a cone.

Example:

```
LIB "gitfan.lib";
intmat Q[3][4] =
1,0,1,0,
0,1,0,1,
0,0,1,1;
ring R = 0,T(1..4),dp;
ideal a = 0;
bigintmat w[1][3] = 3,3,1;
cone lambda = gitCone(a, Q, w);
rays(lambda);
↦ 1,0,0,
↦ 0,1,0,
↦ 1,1,1
```

```
bigintmat w2[1][3] = 1,1,1;
cone lambda2 = gitCone(a, Q, w2);
rays(lambda2);
↦ 1,1,1
```

D.13.4 polymake_lib

Library: polymake.lib

Purpose: Computations with polytopes and fans, interface to TOPCOM

Author: Thomas Markwig, email: keilen@mathematik.uni-kl.de
Yue Ren, email: ren@mathematik.uni-kl.de

Warning: Most procedures will not work unless polymake or topcom is installed and if so, they will only work with the operating system LINUX! For more detailed information see IMPORTANT NOTE respectively consult the help string of the procedures.

The conventions used in this library for polytopes and fans, e.g. the length and labeling of their vertices resp. rays, differs from the conventions used in polymake and thus from the conventions used in the polymake extension polymake.so of Singular. We recommend to use the newer polymake.so whenever possible.

Important note:
Even though this is a Singular library for computing polytopes and fans such as the Newton polytope or the Groebner fan of a polynomial, most of the hard computations are NOT done by Singular but by the program
- TOPCOM by Joerg Rambau, Universitaet Bayreuth (see `http://www.uni-bayreuth.de/de/team/rambau_joerg/TOPCOM/`);
this library should rather be seen as an interface which allows to use a (very limited) number of options which topcom offers to compute with polytopes and fans and to make the results available in Singular for further computations; moreover, the user familiar with Singular does not have to learn the syntax of topcom, if the options offered here are sufficient for his purposes.
Note, though, that the procedures concerned with planar polygons are independent of topcom.

Procedures using topcom: Procedures concerned with planar polygons:

D.13.4.1 triangulations

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: triangulations(polygon[,#]); list polygon, list #

Assume: polygon is a list of integer vectors of the same size representing the affine coordinates of the lattice points

Purpose: the procedure considers the marked polytope given as the convex hull of the lattice points and with these lattice points as markings; it then computes all possible triangulations of this marked polytope

Return: list, each entry corresponds to one triangulation and the ith entry is itself a list of integer vectors of size three, where each integer vector defines one triangle in the triangulation by telling which points of the input are the vertices of the triangle

Note: - the procedure calls for its computations the program points2triangs from the program topcom by Joerg Rambau, Universitaet Bayreuth; it therefore is necessary that this program is installed in order to use this procedure; see http://www.rambau.wm.uni-bayreuth.de/TOPCOM/);
- if you only want to have the regular triangulations the procedure should be called with the string 'regular' as optional argument
- the procedure creates the files /tmp/triangulationsinput and /tmp/triangulationsoutput;
the former is used as input for points2triangs and the latter is its output containing the triangulations of corresponding to points in the format of points2triangs; if you wish to use this for further computations with topcom, you have to call the procedure with the string 'keepfiles' as optional argument
- note that an integer i in an integer vector representing a triangle refers to the ith lattice point, i.e. polygon[i]; this convention is different from TOPCOM's convention, where i would refer to the i-1st lattice point

Example:

```
LIB "polymake.lib";
// the lattice points of the unit square in the plane
list polygon=intvec(0,0),intvec(0,1),intvec(1,0),intvec(1,1);
// the triangulations of this lattice point configuration are computed
list triang=triangulations(polygon);
↦ Evaluating Commandline Options ...
↦ ... done.
↦ 0
↦ 0
triang;
↦ [1]:
↦    [1]:
↦       1,2,3
↦    [2]:
↦       2,3,4
↦ [2]:
↦    [1]:
↦       1,3,4
↦    [2]:
↦       1,2,4
```

D.13.4.2 secondaryPolytope

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: secondaryPolytope(polygon[,#]); list polygon, list #

Assume: - polygon is a list of integer vectors of the same size representing the affine coordinates of lattice points
- if the triangulations of the corresponding polygon have already been computed with the procedure triangulations then these can be given as a second (optional) argument in order to avoid doing this computation again

Purpose: the procedure considers the marked polytope given as the convex hull of the lattice points and with these lattice points as markings; it then computes the lattice points of the secondary polytope given by this marked polytope which correspond to the triangulations computed by the procedure triangulations

Return: list, say L, such that:
L[1] = intmat, each row gives the affine coordinates of a lattice point in the secondary polytope given by the marked
polytope corresponding to polygon
L[2] = the list of corresponding triangulations

Note: if the triangluations are not handed over as optional argument the procedure calls for its computation of these triangulations the program points2triangs from the program topcom by Joerg Rambau, Universitaet Bayreuth; it therefore is necessary that this program is installed in order to use this procedure; see
http://www.rambau.wm.uni-bayreuth.de/TOPCOM/);

Example:

```
LIB "polymake.lib";
// the lattice points of the unit square in the plane
list polygon=intvec(0,0),intvec(0,1),intvec(1,0),intvec(1,1);
// the secondary polytope of this lattice point configuration is computed
list secpoly=secondaryPolytope(polygon);
↦ Evaluating Commandline Options ...
↦ ... done.
↦ 0
↦ 0
// the points in the secondary polytope
print(secpoly[1]);
↦      1     2     2     1
↦      2     1     1     2
// the corresponding triangulations
secpoly[2];
↦ [1]:
↦    [1]:
↦       1,2,3
↦    [2]:
↦       2,3,4
↦ [2]:
↦    [1]:
↦       1,3,4
↦    [2]:
↦       1,2,4
```

D.13.4.3 cycleLength

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: cycleLength(boundary,interior); list boundary, intvec interior

Assume: boundary is a list of integer vectors describing a cycle in some convex lattice polygon around the lattice point interior ordered clock wise

Return: string, the cycle length of the corresponding cycle in the dual tropical curve

Example:

```
LIB "polymake.lib";
// the integer vectors in boundary are lattice points on the boundary
// of a convex lattice polygon in the plane
list boundary=intvec(0,0),intvec(0,1),intvec(0,2),intvec(2,2),
intvec(2,1),intvec(2,0);
// interior is a lattice point in the interior of this lattice polygon
intvec interior=1,1;
// compute the general cycle length of a cycle of the corresponding cycle
// in the dual tropical curve, note that (0,1) and (2,1) do not contribute
cycleLength(boundary,interior);
↦ -4*u11+u00+u02+u22+u20
```

D.13.4.4 splitPolygon

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: splitPolygon (markings); markings list

Assume: markings is a list of integer vectors representing lattice points in the plane which we consider as the marked points of the convex lattice polytope spanned by them

Purpose: split the marked points in the vertices, the points on the facets which are not vertices, and the interior points

Return: list, L consisting of three lists
L[1] : represents the vertices the polygon ordered clockwise
L[1][i][1] = intvec, the coordinates of the ith vertex
L[1][i][2] = int, the position of L[1][i][1] in markings
L[2][i] : represents the lattice points on the facet of the polygon with endpoints L[1][i] and L[1][i+1]
(i considered modulo size(L[1]))
L[2][i][j][1] = intvec, the coordinates of the jth lattice point on that facet
L[2][i][j][2] = int, the position of L[2][i][j][1] in markings
L[3] : represents the interior lattice points of the polygon
L[3][i][1] = intvec, coordinates of ith interior point
L[3][i][2] = int, the position of L[3][i][1] in markings

Example:

```
LIB "polymake.lib";
// the lattice polygon spanned by the points (0,0), (3,0) and (0,3)
// with all integer points as markings
list polygon=intvec(1,1),intvec(3,0),intvec(2,0),intvec(1,0),
intvec(0,0),intvec(2,1),intvec(0,1),intvec(1,2),
intvec(0,2),intvec(0,3);
// split the polygon in its vertices, its facets and its interior points
list sp=splitPolygon(polygon);
// the vertices
sp[1];
↦ [1]:
↦    [1]:
↦       3,0
↦    [2]:
↦       2
↦ [2]:
↦    [1]:
↦       0,0
↦    [2]:
↦       5
↦ [3]:
```

```
↦    [1]:
↦       0,3
↦    [2]:
↦       10
// the points on facets which are not vertices
sp[2];
↦ [1]:
↦    [1]:
↦       [1]:
↦          2,0
↦       [2]:
↦          3
↦    [2]:
↦       [1]:
↦          1,0
↦       [2]:
↦          4
↦ [2]:
↦    [1]:
↦       [1]:
↦          0,1
↦       [2]:
↦          7
↦    [2]:
↦       [1]:
↦          0,2
↦       [2]:
↦          9
↦ [3]:
↦    [1]:
↦       [1]:
↦          1,2
↦       [2]:
↦          8
↦    [2]:
↦       [1]:
↦          2,1
↦       [2]:
↦          6
// the interior points
sp[3];
↦ [1]:
↦    [1]:
↦       1,1
↦    [2]:
↦       1
```

D.13.4.5 eta

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: eta(triang,polygon); triang, polygon list

Assume: polygon has the format of the output of splitPolygon, i.e. it is a list with three entries describing a convex lattice polygon in the following way:
polygon[1] : is a list of lists; for each i the entry polygon[1][i][1] is a lattice point which is a vertex of the lattice
polygon, and polygon[1][i][2] is an integer assigned to this lattice point as identifying index
polygon[2] : is a list of lists; for each vertex of the polygon, i.e. for each entry in polygon[1], it contains a list polygon[2][i], which contains the lattice points on the facet with endpoints polygon[1][i] and polygon[1][i+1] - i considered mod size(polygon[1]); each such lattice point contributes an entry
polygon[2][i][j][1] which is an integer
vector giving the coordinate of the lattice point and an entry polygon[2][i][j][2] which is the identifying index
polygon[3] : is a list of lists, where each entry corresponds to a lattice point in the interior of the polygon, with
polygon[3][j][1] being the coordinates of the point
and polygon[3][j][2] being the identifying index;
triang is a list of integer vectors all of size three describing a triangulation of the polygon described by polygon; if an entry of triang is the vector (i,j,k) then the triangle is built by the vertices with indices i, j and k

Return: intvec, the integer vector eta describing that vertex of the Newton polytope discriminant of the polygon whose dual cone in the Groebner fan contains the cone of the secondary fan of the polygon corresponding to the given triangulation

Note: for a better description of eta see Gelfand, Kapranov,
Zelevinski: Discriminants, Resultants and multidimensional Determinants. Chapter 10.

Example:

```
LIB "polymake.lib";
// the lattice polygon spanned by the points (0,0), (3,0) and (0,3)
// with all integer points as markings
list polygon=intvec(1,1),intvec(3,0),intvec(2,0),intvec(1,0),
intvec(0,0),intvec(2,1),intvec(0,1),intvec(1,2),
intvec(0,2),intvec(0,3);
// split the polygon in its vertices, its facets and its interior points
list sp=splitPolygon(polygon);
// define a triangulation by connecting the only interior point
//       with the vertices
list triang=intvec(1,2,5),intvec(1,5,10),intvec(1,5,10);
// compute the eta-vector of this triangulation
eta(triang,sp);
↦ 9,1,0,0,1,0,0,0,0,1
```

D.13.4.6 findOrientedBoundary

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: findOrientedBoundary(polygon); polygon list

Assume: polygon is a list of integer vectors defining integer lattice points in the plane

Return: list l with the following interpretation
l[1] = list of integer vectors such that the polygonal path defined by these is the

boundary of the convex hull of the lattice points in polygon
l[2] = list, the redundant points in l[1] have been removed

Example:

```
LIB "polymake.lib";
// the following lattice points in the plane define a polygon
list polygon=intvec(0,0),intvec(3,1),intvec(1,0),intvec(2,0),
intvec(1,1),intvec(3,2),intvec(1,2),intvec(2,3),
intvec(2,4);
// we compute its boundary
list boundarypolygon=findOrientedBoundary(polygon);
// the points on the boundary ordered clockwise are boundarypolygon[1]
boundarypolygon[1];
↦ [1]:
↦    0,0
↦ [2]:
↦    1,2
↦ [3]:
↦    2,4
↦ [4]:
↦    3,2
↦ [5]:
↦    3,1
↦ [6]:
↦    2,0
↦ [7]:
↦    1,0
// the vertices of the boundary are boundarypolygon[2]
boundarypolygon[2];
↦ [1]:
↦    0,0
↦ [2]:
↦    2,4
↦ [3]:
↦    3,2
↦ [4]:
↦    3,1
↦ [5]:
↦    2,0
```

D.13.4.7 cyclePoints

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: cyclePoints(triang,points,pt) triang,points list, pt int

Assume: - points is a list of integer vectors describing the lattice points of a marked polygon;
- triang is a list of integer vectors describing a triangulation of the marked polygon in the sense that an integer vector of the form (i,j,k) describes the triangle formed by polygon[i], polygon[j] and polygon[k];
- pt is an integer between 1 and size(points), singling out a lattice point among the marked points

Purpose: consider the convex lattice polygon, say P, spanned by all lattice points in points which in the triangulation triang are connected to the point points[pt]; the procedure computes all marked points in points which lie on the boundary of that polygon, ordered clockwise

Return: list, of integer vectors which are the coordinates of the lattice points on the boundary of the above mentioned polygon P, if this polygon is not the empty set (that would be the case if points[pt] is not a vertex of any triangle in the triangulation); otherwise return the empty list

Example:

```
LIB "polymake.lib";
// the lattice polygon spanned by the points (0,0), (3,0) and (0,3)
// with all integer points as markings
list points=intvec(1,1),intvec(3,0),intvec(2,0),intvec(1,0),
intvec(0,0),intvec(2,1),intvec(0,1),intvec(1,2),
intvec(0,2),intvec(0,3);
// define a triangulation
list triang=intvec(1,2,5),intvec(1,5,7),intvec(1,7,9),intvec(8,9,10),
intvec(1,8,9),intvec(1,2,8);
// compute the points connected to (1,1) in triang
cyclePoints(triang,points,1);
↦ [1]:
↦    3,0
↦ [2]:
↦    0,0
↦ [3]:
↦    0,1
↦ [4]:
↦    0,2
↦ [5]:
↦    1,2
```

D.13.4.8 latticeArea

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: latticeArea(polygon); polygon list

Assume: polygon is a list of integer vectors in the plane

Return: int, the lattice area of the convex hull of the lattice points in polygon, i.e. twice the Euclidean area

Example:

```
LIB "polymake.lib";
// define a polygon with lattice area 5
list polygon=intvec(1,2),intvec(1,0),intvec(2,0),intvec(1,1),
intvec(2,1),intvec(0,0);
latticeArea(polygon);
↦ 5
```

D.13.4.9 picksFormula

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: picksFormula(polygon); polygon list

Assume: polygon is a list of integer vectors in the plane and consider their convex hull C

Return: list, L of three integersthe
L[1] : the lattice area of C, i.e. twice the Euclidean area
L[2] : the number of lattice points on the boundary of C
L[3] : the number of interior lattice points of C

Note: the integers in L are related by Pick's formula, namely: L[1]=L[2]+2*L[3]-2

Example:

```
LIB "polymake.lib";
// define a polygon with lattice area 5
list polygon=intvec(1,2),intvec(1,0),intvec(2,0),intvec(1,1),
intvec(2,1),intvec(0,0);
list pick=picksFormula(polygon);
// the lattice area of the polygon is:
pick[1];
↦ 5
// the number of lattice points on the boundary is:
pick[2];
↦ 5
// the number of interior lattice points is:
pick[3];
↦ 1
// the number's are related by Pick's formula:
pick[1]-pick[2]-2*pick[3]+2;
↦ 0
```

D.13.4.10 ellipticNF

Procedure from library `polymake.lib` (see Section D.13.4 [polymake_lib], page 2128).

Usage: ellipticNF(polygon); polygon list

Assume: polygon is a list of integer vectors in the plane such that their convex hull C has precisely one interior lattice point; i.e. C is the Newton polygon of an elliptic curve

Purpose: compute the normal form of the polygon with respect to the unimodular affine transformations T=A*x+v; there are sixteen different normal forms (see e.g. Bjorn Poonen, Fernando Rodriguez-Villegas: Lattice Polygons and the number 12. Amer. Math. Monthly 107 (2000), no. 3, 238–250.)

Return: list, L such that
L[1] : list whose entries are the vertices of the normal form of the polygon
L[2] : the matrix A of the unimodular transformation
L[3] : the translation vector v of the unimodular transformation
L[4] : list such that the ith entry is the image of polygon[i] under the unimodular transformation T

Example:

```
LIB "polymake.lib";
ring r=0,(x,y),dp;
// the Newton polygon of the following polynomial
//     has precisely one interior point
poly f=x22y11+x19y10+x17y9+x16y9+x12y7+x9y6+x7y5+x2y3;
list polygon=list(intvec(22,11),intvec(19,10),intvec(17,9),
intvec(16,9), intvec(12,7),intvec(9,6),
intvec(7,5),intvec(2,3));
// its lattice points are
polygon;
↦ [1]:
↦    22,11
↦ [2]:
↦    19,10
↦ [3]:
↦    17,9
↦ [4]:
↦    16,9
↦ [5]:
↦    12,7
↦ [6]:
↦    9,6
↦ [7]:
↦    7,5
↦ [8]:
↦    2,3
// find its normal form
list nf=ellipticNF(polygon);
// the vertices of the normal form are
nf[1];
↦ [1]:
↦    4,0
↦ [2]:
↦    0,0
↦ [3]:
↦    0,2
// it has been transformed by the unimodular affine transformation A*x+v
// with matrix A
nf[2];
↦ 3,-7,
↦ -2,5
// and translation vector v
nf[3];
↦ 15,-11
// the 3rd lattice point ...
polygon[3];
↦ 17,9
// ... has been transformed to
nf[4][3];
↦ 3,0
```

D.13.5 realizationMatroids_lib

Library: realizationMatroids.lib

Purpose: Deciding Relative Realizability for Tropical Fan Curves in 2-Dimensional Matroidal Fans

Authors: Anna Lena Winstel, winstel@mathematik.uni-kl.de

Overview: In tropical geometry, one question to ask is the following: given a one-dimensional balanced polyhedral fan C which is set theoretically contained in the tropicalization trop(Y) of an algebraic variety Y, does there exist a curve X in Y such that trop(X) = C? This equality of C and trop(X) denotes an equality of both, the fans trop(X) and C and their weights on the maximal cones. The relative realization space of C with respect to Y is the space of all algebraic curves in Y which tropicalize to C.

This library provides procedures deciding relative realizability for tropical fan curves, i.e. one-dimensional weighted balanced polyhedral fans, contained in two-dimensional matroidal fans trop(Y) where Y is a projective plane.

Notation: If Y is a projective plane in (n-1)-dimensional projective space, we consider trop(Y) in R^n/<1>. Moreover, for the relative realization space of C with respect to Y we only consider algebraic curves of degree deg(C) in Y which tropicalize to C.

Procedures:

D.13.5.1 realizationDim

Procedure from library `realizationMatroids.lib` (see Section D.13.5 [realizationMatroids_lib], page 2137).

Usage: realizationDim(I,C); where I is a homogeneous linear ideal defining the projective plane Y = V(I) and C is a list of intvectors such that each intvector represents a one-dimensional cone in the tropical fan curve whose relative realizability should be checked. This representation is done in the following way: the one-dimensional cone K is represented by a vector w whose equivalence class [w] in R^n/<1> can be written as [w] = m*[v] where [v] is the primitive generator of K and m is the weight of K.

Returns: the dimension of the relative realization space of the tropical curve C with respect to Y, and -1 if the relative realization space is empty.

Example:

```
LIB "realizationMatroids.lib";
ring r = 0,(x(1..4)),dp;
ideal I = x(1)+x(2)+x(3)+x(4);
list C = list(intvec(2,2,0,0),intvec(0,0,2,1),intvec(0,0,0,1));
//C represents the tropical fan curve which consists of the cones
//cone([(1,1,0,0)]) (with weight 2), cone([(0,0,2,1)]) (with weight 1)
//and cone([(0,0,0,1)]) (with weight 1)
realizationDim(I,C);
↦ -1
```

D.13.5.2 irrRealizationDim

Procedure from library `realizationMatroids.lib` (see Section D.13.5 [realizationMatroids_lib], page 2137).

Usage: irrRealizationDim(I,C); where I is a homogeneous linear ideal defining the projective plane Y = V(I) and C is a list of intvectors such that each intvector represents a one-dimensional cone in the tropical fan curve whose irreducible relative realizability should be checked. This representation is done in the following way: a one-dimensional cone K is represented by a vector w whose equivalence class [w] in R^n/<1> can be written as [w] = m*[v] where [v] is the primitive generator of K and m is the weight of K.

Returns: the dimension of the irreducible relative realization space of C with respect to Y, and -1 if the irreducible realization space is empty.

Example:

```
LIB "realizationMatroids.lib";
ring r = 0,(x(1..4)),dp;
ideal I = x(1)+x(2)+x(3)+x(4);
list C = list(intvec(2,2,0,0),intvec(0,0,2,2));
//C represents the tropical fan curve which consists of the cones
//cone([(1,1,0,0)]) and cone([(1,1,0,0)]), both with weight 2
realizationDim(I,C);
↦ 0
irrRealizationDim(I,C);
↦ -1
```

D.13.5.3 realizationDimPoly

Procedure from library `realizationMatroids.lib` (see Section D.13.5 [realizationMatroids_lib], page 2137).

Usage: realizationDimPoly(I,C); where I is a homogeneous linear ideal defining the projective plane Y = V(I) and C is a list of intvectors such that each intvector represents a one-dimensional cone in the tropical fan curve whose relative realizability should be checked. This representation is done in the following way: the one-dimensional cone K is represented by a vector w whose equivalence class [w] in R^n/<1> can be written as [w] = m*[v] where [v] is the primitive generator of K and m is the weight of K.

Returns: If the relative realization space of the tropical fan curve C is non-empty, this routine returns the tuple (r,f), where r is the dimension of the relative realization space and f is an example of a homogeneous polynomial of degree deg(C) cutting out a curve X in Y which tropicalizes to C. In case the relative realization space is empty, the output is set to -1.

Example:

```
LIB "realizationMatroids.lib";
ring r = 0,(x(1..4)),dp;
ideal I = x(1)+x(2)+x(3)+x(4);
list C = list(intvec(2,2,0,0),intvec(0,0,2,2));
//C represents the tropical fan curve which consists of the cones
//cone([(1,1,0,0)]) and cone([(1,1,0,0)]), both with weight 2
realizationDimPoly(I,C);
↦ 0 x(1)^2+2*x(1)*x(2)+x(2)^2
C = list(intvec(0,0,0,4),intvec(0,1,3,0),intvec(1,0,1,0),intvec(0,2,0,0),intvec(3,1,0
//C represents the tropical fan curve which consists of the cones
//cone([(0,0,0,1)]) with weight 4,
//cone([(0,1,3,0)]), cone([(1,0,1,0)]) both with weight 1,
//cone([(0,1,0,0)]) with weight 2, and
//cone([(3,1,0,0)]) with weight 1
realizationDimPoly(I,C);
↦ 7 x(1)*x(2)^3+x(1)^3*x(3)+x(2)^3*x(3)+x(1)*x(3)^3
```

D.13.6 tropical_lib

Library: tropical.lib

Purpose: Computations in Tropical Geometry

Authors: Anders Jensen Needergard, email: jensen@math.tu-berlin.de
Hannah Markwig, email: hannah@uni-math.gwdg.de
Thomas Markwig, email: keilen@mathematik.uni-kl.de
Yue Ren, email: ren@mathematik.uni-kl.de

Warning: - tropicalLifting will only work with LINUX and if in addition gfan is installed.
- drawTropicalCurve and drawTropicalNewtonSubdivision will only display the tropical curve with LINUX and if in addition latex and xdg-open
are installed.
- For tropicalLifting in the definition of the basering the parameter t
from the Puiseux series field C{{t}} must be defined as a variable,
while for all other procedures it must be defined as a parameter.

Theory: Fix some base field K and a bunch of lattice points v0,...,vm in the integer lattice Z^n, then this defines a toric variety as the closure of (K*)^n in the projective space P^m, where the torus is embedded via the map sending a point x in (K*)^n to the point (x^v0,...,x^vm).
The generic hyperplane sections are just the images of the hypersurfaces in (K*)^n defined by the polynomials f=a0*x^v0+...+am*x^vm=0. Some properties of these hypersurfaces can be studied via tropicalisation.

For this we suppose that K=C{{t}} is the field of Puiseux series over the field of complex numbers (or any other field with a valuation into the real numbers). One associates to the hypersurface given by f=a0*x^v0+...+am*x^vm the tropical hypersurface defined by the tropicalisation trop(f)=min{val(a0)+<v0,x>,...,val(am)+<vm,x>}.
Here, <v,x> denotes the standard scalar product of the integer vector v in Z^n with the vector x=(x1,...,xn) of variables, so that trop(f) is a piecewise linear function on R^n.
The corner locus of this function (i.e. the points at which the minimum is attained a least twice) is the tropical hypersurface defined by trop(f).
The theorem of Newton-Kapranov states that this tropical hypersurface is the same as if one computes pointwise the valuation of the hypersurface given by f. The analogue holds true if one replaces one equation f by an ideal I. A constructive proof of the theorem is given by an adapted version of the Newton-Puiseux algorithm. The hard part is to find a point in the variety over C{{t}} which corresponds to a given point in the tropical variety.

It is the purpose of this library to provide basic means to deal with tropical varieties. Of course we cannot represent the field of Puiseux series over C in its full strength, however, in order to compute interesting examples it will be sufficient to replace the complex numbers C by the rational numbers Q and to replace Puiseux series in t by rational functions in t, i.e. we replace C{{t}} by Q(t), or sometimes even by Q[t]. Note, that this in particular forbids rational exponents for the t's.

Moreover, in Singular no negative exponents of monomials are allowed, so that the integer vectors vi will have to have non-negative entries. Shifting all exponents by a fixed integer vector does not change the tropicalisation nor does it change the toric variety. Thus this does not cause any restriction.
If, however, for some reason you prefer to work with general vi, then you have to pass right away to the tropicalisation of the equations, wherever this is allowed – these are linear polynomials where the constant coefficient corresponds to the valuation of the original coefficient and where the non-constant coefficient correspond to the exponents of the monomials, thus they may be rational numbers respectively negative numbers: e.g. if f=t^{1/2}*x^{-2}*y^3+2t*x*y+4 then trop(f)=min{1/2-2x+3y,1+x+y,0}.

The main tools provided in this library are as follows:
- tropicalLifting implements the constructive proof of the Theorem of Newton-Kapranov and constructs a point in the variety over C{{t}} corresponding to a given point in the
corresponding tropical variety associated to an
ideal I; the generators of I have to be in the
polynomial ring Q[t,x1,...,xn] considered as a
subring of C{{t}}[x1,...,xn]; a solution will be
constructed up to given order; note that several
field extensions of Q might be necessary throughout
the intermediate computations; the procedures use
the external program gfan
- puiseuxExpansion computes a Newton-Puiseux expansion of a plane curve singularity
- drawTropicalCurve visualises a tropical plane curve either given by a polynomial in Q(t)[x,y] or by a list of linear
polynomials of the form ax+by+c with a,b in Z and c
in Q; latex must be installed on your computer
- tropicalJInvariant computes the tropical j-invaiant of a tropical elliptic curve

Procedures for tropical lifting: Procedures for drawing tropical curves: General procedures: Procedures for latex conversion: Auxiliary procedures: Procedures from binary library:

D.13.6.1 tropicalLifting

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicalLifting(i,w,ord[,opt]); i ideal, w intvec, ord int, opt string

Assume: - i is an ideal in Q[t,x_1,...,x_n], w=(w_0,w_1,...,w_n) and (w_1/w_0,...,w_n/w_0) is in the tropical variety of i, and ord is the order up to which a point in V(i) over Q{{t}} lying over (w_1/w_0,...,w_n/w_0) shall be computed;
w_0 may NOT be ZERO
- the basering should not have any parameters on its own and it should have a global monomial ordering,
e.g. ring r=0,(t,x(1..n)),dp;
- the first variable of the basering will be treated as the parameter t in the Puiseux series field
- the optional parameter opt should be one or more strings among the following:
'isZeroDimensional' : the dimension i is zero (not to be checked);
'isPrime' : the ideal is prime over Q(t)[x_1,...,x_n] (not to be checked);
'isInTrop' : (w_1/w_0,...,w_n/w_0) is in the tropical variety (not to be checked);
'oldGfan' : uses gfan version 0.2.1 or less
'findAll' : find all solutions of a zero-dimensional ideal over (w_1/w_0,...,w_n/w_0)
'noAbs' : do NOT use absolute primary decomposition
'puiseux' : n=1 and i is generated by one equation
'noResubst' : avoids the computation of the resubstitution

Return: IF THE OPTION 'findAll' WAS NOT SET THEN:
list, containing one lifting of the given point (w_1/w_0,...,w_n/w_0) in the tropical variety of i to a point in V(i) over Puiseux series field up to the first ord terms; more precisely:
IF THE OPTION 'noAbs' WAS NOT SET, THEN:

l[1] = ring Q[a]/m[[t]]
l[2] = int
l[3] = intvec
l[4] = list
IF THE OPTION 'noAbs' WAS SET, THEN:
l[1] = ring Q[X(1),...,X(k)]/m[[t]]
l[2] = int
l[3] = intvec
l[4] = list
l[5] = string
IF THE OPTION 'findAll' WAS SET, THEN:
list, containing ALL liftings of the given point ((w_1/w_0,...,w_n/w_0) in the tropical variety of i to a point in V(i) over Puiseux series field up to the first ord terms, if the ideal is zero-dimensional over Q{{t}};
more precisely, each entry of the list is a list l as computed if 'findAll' was NOT set
WE NOW DESCRIBE THE LIST ENTRIES IF 'findAll' WAS NOT SET:
- the ring l[1] contains an ideal LIFT, which contains a point in V(i) lying over w up to the first ord terms;
- and if the integer l[2] is N then t has to be replaced by t^1/N in the lift, or alternatively replace t by t^N in the defining ideal
- if the k+1st entry of l[3] is non-zero, then the kth component of LIFT has to be multiplied t^(-l[3][k]/l[3][1]) AFTER substituting t by t^1/N
- unless the option 'noResubst' was set, the kth entry of list l[4] is a string which represents the kth generator of
the ideal i where the coordinates have been replaced by the result of the lift;
the t-order of the kth entry should in principle be larger than the t-degree of LIFT
- if the option 'noAbs' was set, then the string in l[5] defines a maximal ideal in the field Q[X(1),...,X(k)], where X(1),...,X(k) are the parameters of the ring in l[1];
the basefield of the ring in l[1] should be considered modulo this ideal

Remark: - it is best to use the procedure displayTropicalLifting to display the result
- the option 'findAll' cannot be used if 'noAbs' is set
- if the parameter 'findAll' is set AND the ideal i is zero-dimensional in Q{{t}}[x_1,...,x_n] then ALL points in V(i) lying over w are computed up to order ord; if the ideal is not-zero dimenisonal, then only the points in the ideal after cutting down to dimension zero will be computed
- the procedure requires that the program GFAN is installed on your computer; if you have GFAN version less than 0.3.0 then you must use the optional parameter 'oldGfan'
- the procedure requires the Singular procedure absPrimdecGTZ to be present in the package primdec.lib, unless the option 'noAbs' is set; but even if absPrimdecGTZ is present it might be necessary to set the option 'noAbs' in order to avoid the costly absolute primary decomposition; the side effect is that the field extension which is computed throughout the recursion might need more than one parameter to be described
- since Q is infinite, the procedure finishes with probability one
- you can call the procedure with Z/pZ as base field instead of Q, but there are some problems you should be aware of:
+ the Puiseux series field over the algebraic closure of Z/pZ is NOT algebraicall closed, and thus there may not exist a point in V(i) over the Puiseux series field with the desired valuation; so there is no chance that the procedure produced a sensible

output - e.g. if i=tx^p-tx-1
+ if the dimension of i over Z/pZ(t) is not zero the process of reduction to zero might not work if the characteristic is small and you are unlucky
+ the option 'noAbs' has to be used since absolute primary decomposition in Singular only works in characteristic zero
- the basefield should either be Q or Z/pZ for some prime p; field extensions will be computed if necessary; if you need parameters or field extensions from the beginning they should rather be simulated as variables possibly adding their relations to the ideal; the weights for the additional variables should be zero

Example:

```
LIB "tropical.lib";
int oldprintlevel=printlevel;
printlevel=1;
////////////////////////////////////////////////////////
// 1st EXAMPLE:
////////////////////////////////////////////////////////
ring r=0,(t,x),dp;
ideal i=(1+t2)*x2+t5x+t2;
intvec w=1,-1;
list LIST=tropicalLifting(i,w,4);
↦ // ** name conflict var(1) and var(3): `x(1)`, rename to `@x(1)`in >>    \
     ring EXTENSIONRING = ring(RL);<<
↦ in tropical.lib::findzerosAndBasictransform:6332
↦ LP algorithm being used: "cddgmp".
↦ Groebner basis Engine being used: "gfan".
↦ LP algorithm being used: "cddgmp".
↦ Groebner basis Engine being used: "gfan".
↦ LP algorithm being used: "cddgmp".
↦ Groebner basis Engine being used: "gfan".
↦ 0
↦ 0
↦ 0
↦ 0
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
   ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
   LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦ 
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦ 
↦ def LIFTRing=L[1]; setring LIFTRing; LIFT;
↦ 
def LIFTRing=LIST[1];
setring LIFTRing;
// LIFT contains the first 4 terms of a point in the variety of i
// over the Puiseux series field C{{t}} whose order is -w[1]/w[0]=1
LIFT;
↦ LIFT[1]=(a)*t+(-a)/2*t3+(3a-4)/8*t5+(-5a+8)/16*t7
```

```
// Since the computations were done over Q a field extension was necessary,
// and the parameter "a" satisfies the equation given by minpoly
minpoly;
↦ 0
////////////////////////////////////////////////////////
// 2nd EXAMPLE
////////////////////////////////////////////////////////
setring r;
LIST=tropicalLifting(x12-t11,intvec(12,-11),2,"isPrime","isInTrop");
↦ 0
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
   ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
   LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦ 
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦ 
↦ def LIFTRing=L[1]; setring LIFTRing; LIFT;
↦ 
def LIFTRing2=LIST[1];
setring LIFTRing2;
// This time, LIFT contains the lifting of the point -w[1]/w[0]=11/12
// only after we replace in LIFT the variable t by t^1/N with N=LIST[3]
LIFT;
↦ LIFT[1]=t11
LIST[3];
↦ 12,0
///////////////////////////////////////////////////////
// 3rd EXAMPLE
////////////////////////////////////////////////////////
ring R=0,(t,x,y,z),dp;
ideal i=-y2t4+x2,yt3+xz+y;
w=1,-2,0,2;
LIST=tropicalLifting(i,w,3);
↦ LP algorithm being used: "cddgmp".
↦ Groebner basis Engine being used: "gfan".
↦ 0
↦ 0
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
   ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
   LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦ 
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦ 
```

```
↦ def LIFTRing=L[1]; setring LIFTRing; LIFT;
↦
// This time, LIFT contains the lifting of the point v=(-2,0,2)
// only after we multiply LIFT[3] by t^k with k=-LIST[4][3];
// NOTE: since the last component of v is positive, the lifting
//       must start with a negative power of t, which in Singular
//       is not allowed for a variable.
def LIFTRing3=LIST[1];
setring LIFTRing3;
LIFT;
↦ LIFT[1]=t2
↦ LIFT[2]=1
↦ LIFT[3]=-1-t3
LIST[4];
↦ [1]:
↦    0
↦ [2]:
↦    0
// An easier way to display this is via displayTropicalLifting.
setring R;
displayTropicalLifting(LIST,"subst");
↦ The lifting of the point in the tropical variety lives in the ring
↦ Q[[t]]
↦
↦ The lifting has the form:
↦ x=(1)*t^2
↦ y=(1)
↦ z=(-1)*1/t^2 + (-1)*t
↦
↦ Substituting the solution into the ideal gives:
↦ i[1]=0
↦ i[2]=0
printlevel=oldprintlevel;
```

D.13.6.2 displayTropicalLifting

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: displaytropcallifting(troplift[,#]); troplift list, # list

Assume: troplift is the output of tropicalLifting; the optional parameter # can be the string 'subst'

Return: none

Note: - the procedure displays the output of the procedure tropicalLifting
- if the optional parameter 'subst' is given, then the lifting is substituted into the ideal and the result is displayed

Example:

```
LIB "tropical.lib";
ring r=0,(t,x,y,z),dp;
ideal i=-y2t4+x2,yt3+xz+y;
intvec w=2,-4,0,4;
displayTropicalLifting(tropicalLifting(i,w,3),"subst");
```

```
↦ LP algorithm being used: "cddgmp".
↦ Groebner basis Engine being used: "gfan".
↦ 0
↦ 0
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
  ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
  LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦
↦ def LIFTRing=L[1]; setring LIFTRing; LIFT;
↦
↦ The lifting of the point in the tropical variety lives in the ring
↦ Q[[t^(1/2)]]
↦
↦ The lifting has the form:
↦ x=(1)*t^(4/2)
↦ y=(1)
↦ z=(-1)*1/t^(4/2) + (-1)*t^(2/2)
↦
↦ Substituting the solution into the ideal gives:
↦ i[1]=0
↦ i[2]=0
```

D.13.6.3 puiseuxExpansion

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: puiseuxExpansion(f,n,#); f poly, n int, # list

Assume: f is a non-constant polynomial in two variables which is not divisible by the first variable and which is squarefree as a power series over the complex numbers;
the base field is either the field of rational numbers or a finite extension thereof; monomial ordering is assumed to be local;
the optional parameter # can be the string 'subst'

Return: list, where each entry of the list l describes the Newton-Puiseux
parametrisations of one branch of the plane curve singularity
at the origin defined by f; only the terms up to order n of each
parametetrisation are computed
l[i][1] = is a ring
l[i][2] = int
l[i][3] = string

WE NOW DESCRIBE THE LIST ENTRIES l[i] IN MORE DETAIL:
- the ring l[i][1] contains an ideal LIFT and the Newton-Puiseux parametrisation of the branch is given by x=t^N and y=LIFT[1], where N=l[i][2]
- if the base field had a parameter and a minimal polynomial, then the new base field will have a parameter and a new minimal polynomial, and LIFT[2] describes how the

old parameter can be computed from the new one
- if a field extension with minimal polynomial of degree k was necessary, then to the one extension produced actually k extensions correspond by replacing the parameter a successively by all zeros of the minimal polynomial
- if the option subst was set l[i][3] contains the polynomial where y has been substituted by y(t^{1/N}) as a string

Remark: - it is best to use the procedure displayPuiseuxExpansion to display the result
- the procedure requires the Singular procedure absPrimdecGTZ to be present in the package primdec.lib
- if f is not squarefree it will be replaced by its squarefree part

Example:

```
LIB "tropical.lib";
printlevel=1;
ring r=0,(x,y),ds;
poly f=x2-y4+x5y7;
puiseuxExpansion(f,3,"subst");
↦ // ** name conflict var(1) and var(3): ‘x(1)‘, rename to ‘@x(1)‘in >>    \
     ring EXTENSIONRING = ring(RL);<<
↦ in tropical.lib::findzerosAndBasictransform:6332
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
   );) tropical.lib::tropicalparametrise:5778
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
   );) tropical.lib::tropicalparametrise:5778
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
   ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
   LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦ 
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦ 
↦ def LIFTRing=L[1][1]; setring LIFTRing; LIFT;
↦ 
↦ !!!! WARNING: The number of terms computed in the Puiseux expansion were
↦ !!!!          not enough to find all branches of the curve singularity!
↦ [1]:
↦    [1]:
↦       // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering ls
↦ //                  : names    t
↦ //        block   2 : ordering C
↦    [2]:
↦       2
↦    [3]:
↦       (11/8)*t^(30/2) + (5/4)*t^(43/2) + (139/256)*t^(56/2) + (35/256)*t^\
   (69/2) + (21/1024)*t^(82/2) + (7/4096)*t^(95/2) + (1/16384)*t^(108/2)
↦ [2]:
```

```
↦    [1]:
↦       // coefficients: QQ
↦ // number of vars : 1
↦ //        block   1 : ordering ls
↦ //                  : names    t
↦ //        block   2 : ordering C
↦    [2]:
↦       2
↦    [3]:
↦       (11/8)*t^(30/2) + (-5/4)*t^(43/2) + (139/256)*t^(56/2) + (-35/256)*\
   t^(69/2) + (21/1024)*t^(82/2) + (-7/4096)*t^(95/2) + (1/16384)*t^(108/2)
↦ [3]:
↦    [1]:
↦       // coefficients: QQ(a)
↦ // number of vars : 1
↦ //        block   1 : ordering ls
↦ //                  : names    t
↦ //        block   2 : ordering C
↦    [2]:
↦       2
↦    [3]:
↦       (-a4+1)*t^(4/2) + (a7-a3)*t^(17/2) + (14a6-3a2)/8*t^(30/2) + (21a5-\
   a)/16*t^(43/2) + (140a4-1)/256*t^(56/2) + (35a3)/256*t^(69/2) + (21a2)/10\
   24*t^(82/2) + (7a)/4096*t^(95/2) + (1/16384)*t^(108/2)
displayPuiseuxExpansion(puiseuxExpansion(f,3));
↦ // ** name conflict var(1) and var(3): `x(1)`, rename to `@x(1)`in >>    \
     ring EXTENSIONRING = ring(RL);<<
↦ in tropical.lib::findzerosAndBasictransform:6332
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
   );) tropical.lib::tropicalparametrise:5778
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
   );) tropical.lib::tropicalparametrise:5778
↦ The procedure has created a list of lists. The jth entry of this list
↦ contains a ring, an integer and an intvec.
↦ In this ring lives an ideal representing the wanted lifting,
↦ if the integer is N then in the parametrisation t has to be replaced by t\
   ^1/N,
↦ and if the ith component of the intvec is w[i] then the ith component in \
   LIFT
↦ should be multiplied by t^-w[i]/N in order to get the parametrisation.
↦
↦ Suppose your list has the name L, then you can access the 1st ring via:
↦
↦ def LIFTRing=L[1][1]; setring LIFTRing; LIFT;
↦
↦ !!!! WARNING: The number of terms computed in the Puiseux expansion were
↦ !!!!          not enough to find all branches of the curve singularity!
↦ ============================
↦ 1. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[[t^(1/2)]]
↦
```

```
↦ The expansion has the form:
↦ y=(1)*t^(1/2) + (1/4)*t^(14/2)
↦
↦ =============================
↦ 2. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[[t^(1/2)]]
↦
↦ The expansion has the form:
↦ y=(-1)*t^(1/2) + (1/4)*t^(14/2)
↦
↦ =============================
↦ 3. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[a]/0[[t^(1/2)]]
↦
↦ The expansion has the form:
↦ y=(a)*t^(1/2) + (1/4)*t^(14/2)
↦
```

D.13.6.4 displayPuiseuxExpansion

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: displayPuiseuxExpansion(puiseux[,#]); puiseux list, # list

Assume: puiseux is the output of puiseuxExpansion; the optional parameter # can be the string 'subst'

Return: none

Note: - the procedure displays the output of the procedure puiseuxExpansion
- if the optional parameter 'subst' is given, then the expansion is substituted into the polynomial and the result is displayed
- if the base field had a parameter and a minimal polynomial, then the new base field will have a parameter and a minimal polynomial; var(2) is the old parameter and it is displayed how the old parameter can be computed from the new one

Example:

```
LIB "tropical.lib";
ring r=0,(x,y),ds;
poly f=x2-y4+x5y7;
displayPuiseuxExpansion(puiseuxExpansion(f,3));
↦ // ** name conflict var(1) and var(3): ‘x(1)‘, rename to ‘@x(1)‘in >>    \
   ring EXTENSIONRING = ring(RL);<<
↦ in tropical.lib::findzerosAndBasictransform:6332
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
);) tropical.lib::tropicalparametrise:5778
↦ // ** redefining ggteiler (              int ggteiler=gcd(wneu[1],wneu[2]\
);) tropical.lib::tropicalparametrise:5778
↦ !!!! WARNING: The number of terms computed in the Puiseux expansion were
↦ !!!!          not enough to find all branches of the curve singularity!
↦ =============================
↦ 1. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[[t^(1/2)]]
↦
↦ The expansion has the form:
↦ y=(1)*t^(1/2) + (1/4)*t^(14/2)
↦
↦ =============================
↦ 2. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[[t^(1/2)]]
↦
↦ The expansion has the form:
↦ y=(-1)*t^(1/2) + (1/4)*t^(14/2)
↦
↦ =============================
↦ 3. Expansion:
↦
↦ The Puiseux expansion lives in the ring
↦ Q[a]/0[[t^(1/2)]]
↦
↦ The expansion has the form:
↦ y=(a)*t^(1/2) + (1/4)*t^(14/2)
↦
```

D.13.6.5 tropicalCurve

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicalCurve(tp[,#]); tp list, # optional list

Assume: tp is list of linear polynomials of the form ax+by+c
with integers a, b and a rational number c representing a tropical Laurent polynomial defining a tropical plane curve; alternatively tp can be a polynomial in Q(t)[x,y] defining a tropical plane curve via the valuation map;
the basering must have a global monomial ordering,
two variables and up to one parameter!

Return: list, each entry i=1,...,size(l)-1 corresponds to a vertex in the tropical plane curve defined by tp
l[i][1] = x-coordinate of the ith vertex
l[i][2] = y-coordinate of the ith vertex
l[i][3] = intmat, if j is an entry in the first row
of intmat then the ith vertex of
the tropical curve is connected to the
jth vertex with multiplicity given
by the corresponding entry in the second row
l[i][4] = list of lists, the first entry of a list is a primitive integer vector defining the direction
of an unbounded edge emerging from the ith vertex
of the graph, the corresponding second entry in
the list is the multiplicity of the unbounded edge

l[i][5] = a polynomial whose monomials mark the vertices in the Newton polygon corresponding to the entries
in tp which take the common minimum at the ith
vertex – if some coefficient a or b of the
linear polynomials in the input was negative,
then each monomial has to be shifted by
the values in l[size(l)][3]
l[size(l)][1] = list, the entries describe the boundary points of the Newton subdivision
l[size(l)][2] = list, the entries are pairs of integer vectors defining an interior
edge of the Newton subdivision
l[size(l)][3] = intvec, the monomials occuring in l[i][5] have to be shifted by this vector
in order to represent marked
vertices in the Newton polygon

Note: here the tropical polynomial is supposed to be the MINIMUM of the linear forms in tp, unless the optional input #[1] is the string 'max'

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t*(x7+y7+1)+1/t*(x4+y4+x2+y2+x3y+xy3)+1/t7*x2y2;
list graph=tropicalCurve(f);
// the tropical curve has size(graph)-1 vertices
size(graph)-1;
↦ 7
// the coordinates of the first vertex are graph[1][1],graph[1][2];
graph[1][1],graph[1][2];
↦ -8/3 -8/3
// the first vertex is connected to the vertices
//     graph[1][3][1,1..ncols(graph[1][3])]
intmat M=graph[1][3];
M[1,1..ncols(graph[1][3])];
↦ 2 3
// the weights of the edges to these vertices are
//     graph[1][3][2,1..ncols(graph[1][3])]
M[2,1..ncols(graph[1][3])];
↦ 1 1
// from the first vertex emerge size(graph[1][4]) unbounded edges
size(graph[1][4]);
↦ 1
// the primitive integral direction vector of the first unbounded edge
//     of the first vertex
graph[1][4][1][1];
↦ -1,-1
// the weight of the first unbounded edge of the first vertex
graph[1][4][1][2];
↦ 7
// the monomials which are part of the Newton subdivision of the first vertex
graph[1][5];
↦ x7+y7+x2y2
// connecting the points in graph[size(graph)][1] we get
//     the boundary of the Newton polytope
graph[size(graph)][1];
```

```
↦ [1]:
↦    0,7
↦ [2]:
↦    7,0
↦ [3]:
↦    0,0
// an entry in graph[size(graph)][2] is a pair of points
//     in the Newton polytope bounding an inner edge
graph[size(graph)][2][1];
↦ [1]:
↦    7,0
↦ [2]:
↦    2,2
```

D.13.6.6 drawTropicalCurve

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: drawTropicalCurve(f[,#]); f poly or list, # optional list

Assume: f is list of linear polynomials of the form ax+by+c with integers a, b and a rational number c representing a tropical Laurent polynomial defining a tropical plane curve; alternatively f can be a polynomial in Q(t)[x,y] defining a tropical plane curve via the valuation map;
the basering must have a global monomial ordering, two variables and up to one parameter!

Return: NONE

Note: - the procedure creates the files /tmp/tropicalcurveNUMBER.tex and /tmp/tropicalcurveNUMBER.ps, where NUMBER is a random four digit integer;
moreover it displays the tropical curve via xdg-open;
if you wish to remove all these files from /tmp,
call the procedure cleanTmp
- edges with multiplicity greater than one carry this multiplicity
- if # is empty, then the tropical curve is computed w.r.t. minimum, if #[1] is the string 'max', then it is computed w.r.t. maximum
- if the last optional argument is 'onlytexfile' then only the latex file is produced; this option should be used if xdg-utils is not installed on your system
- note that lattice points in the Newton subdivision which are black correspond to markings of the marked subdivision, while lattice points in grey are not marked

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t*(x3+y3+1)+1/t*(x2+y2+x+y+x2y+xy2)+1/t2*xy;
// the command drawTropicalCurve(f) computes the graph of the tropical curve
// given by f and displays a post script image, provided you have xdg-open
drawTropicalCurve(f);
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (/tmp/tropicalcurve6034.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/amscls/amsart.cls
↦ Document Class: amsart 2015/03/04 v2.20.2
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦
↦ ! LaTeX Error: File ‘texdraw.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name: warning: kpathsea: $.sty: Unrecognized variable construc\
   t ‘$.’.
↦
↦ ! LaTeX Error: File ‘$.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name:
↦ ! Emergency stop.
↦ <read *>
↦
↦ l.3 \setlength
↦                {\topmargin}{30mm}^^M
↦ No pages of output.
↦ Transcript written on tropicalcurve6034.log.
↦ This is dvips(k) 5.997 Copyright 2017 Radical Eye Software (www.radicaley\
   e.com)
↦ dvips: DVI file can’t be opened: /tmp/tropicalcurve6034.dvi: No such file\
    or directory
↦ rm: das Entfernen von ’tropicalcurve6034.aux’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’tropicalcurve6034.ps?’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’tropicalcurve6034.dvi’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ sh: xdg-open: Kommando nicht gefunden.
↦ 0
// we can instead apply the procedure to a tropical polynomial and use "maximum"
poly g=1/t3*(x7+y7+1)+t3*(x4+y4+x2+y2+x3y+xy3)+t21*x2y2;
list tropical_g=tropicalise(g);
tropical_g;
↦ [1]:
↦    7*x-3
↦ [2]:
```

```
↦    7*y-3
↦ [3]:
↦    4*x+3
↦ [4]:
↦    3*x+y+3
↦ [5]:
↦    2*x+2*y+21
↦ [6]:
↦    x+3*y+3
↦ [7]:
↦    4*y+3
↦ [8]:
↦    2*x+3
↦ [9]:
↦    2*y+3
↦ [10]:
↦    -3
drawTropicalCurve(tropical_g,"max");
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (/tmp/tropicalcurve9641.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/amscls/amsart.cls
↦ Document Class: amsart 2015/03/04 v2.20.2
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦
↦ ! LaTeX Error: File ‘texdraw.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name:
↦ ! Emergency stop.
↦ <read *>
↦
↦ l.3 \setlength
↦               {\topmargin}{30mm}^^M
↦ No pages of output.
↦ Transcript written on tropicalcurve9641.log.
↦ This is dvips(k) 5.997 Copyright 2017 Radical Eye Software (www.radicaley\
  e.com)
↦ dvips: DVI file can’t be opened: /tmp/tropicalcurve9641.dvi: No such file\
   or directory
```

```
↦ rm: das Entfernen von ’tropicalcurve9641.aux’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’tropicalcurve9641.ps?’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’tropicalcurve9641.dvi’ ist nicht mglich: Datei o\
   der Verzeichnis nicht gefunden
↦ 0
↦ sh: xdg-open: Kommando nicht gefunden.
```

D.13.6.7 drawNewtonSubdivision

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: drawTropicalCurve(f[,#]); f poly, # optional list

Assume: f is list of linear polynomials of the form ax+by+c with integers a, b and a rational number c representing a tropical Laurent polynomial defining a tropical plane curve;
alternatively f can be a polynomial in Q(t)[x,y] defining a tropical plane curve via the valuation map;
the basering must have a global monomial ordering, two variables and up to one parameter!

Return: NONE

Note: - the procedure creates the files /tmp/newtonsubdivisionNUMBER.tex, and /tmp/newtonsubdivisionNUMBER.ps, where NUMBER is a random four digit integer;
moreover it desplays the tropical curve defined by f via xdg-open; if you wish to remove all these files from /tmp, call the procedure cleanTmp;
if # is empty, then the tropical curve is computed w.r.t. minimum, if #[1] is the string 'max', then it is computed w.r.t. maximum
- note that lattice points in the Newton subdivision which are black correspond to markings of the marked subdivision, while lattice points in grey are not marked

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t*(x3+y3+1)+1/t*(x2+y2+x+y+x2y+xy2)+1/t2*xy;
// the command drawTropicalCurve(f) computes the graph of the tropical curve
// given by f and displays a post script image, provided you have xdg-open
drawNewtonSubdivision(f);
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
    format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (/tmp/newtonsubdivision6034.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/amscls/amsart.cls
↦ Document Class: amsart 2015/03/04 v2.20.2
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦
↦ ! LaTeX Error: File ‘texdraw.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name: warning: kpathsea: $.sty: Unrecognized variable construc\
  t ‘$.’.
↦
↦ ! LaTeX Error: File ‘$.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name:
↦ ! Emergency stop.
↦ <read *>
↦
↦ l.3 \begin
↦           {document}^^M
↦ No pages of output.
↦ Transcript written on newtonsubdivision6034.log.
↦ This is dvips(k) 5.997 Copyright 2017 Radical Eye Software (www.radicaley\
  e.com)
↦ dvips: DVI file can’t be opened: /tmp/newtonsubdivision6034.dvi: No such \
  file or directory
↦ rm: das Entfernen von ’newtonsubdivision6034.aux’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’newtonsubdivision6034.ps?’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’newtonsubdivision6034.dvi’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ 0
↦ sh: xdg-open: Kommando nicht gefunden.
// we can instead apply the procedure to a tropical polynomial
poly g=x+y+x2y+xy2+1/t*xy;
list tropical_g=tropicalise(g);
tropical_g;
↦ [1]:
↦    2*x+y
↦ [2]:
↦    x+2*y
↦ [3]:
↦    x+y-1
↦ [4]:
↦    x
↦ [5]:
↦    y
drawNewtonSubdivision(tropical_g);
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (/tmp/newtonsubdivision9641.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/amscls/amsart.cls
↦ Document Class: amsart 2015/03/04 v2.20.2
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦
↦ ! LaTeX Error: File ‘texdraw.sty’ not found.
↦
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦
↦ Enter file name:
↦ ! Emergency stop.
↦ <read *>
↦
↦ l.3 \begin
↦           {document}^^M
↦ No pages of output.
↦ Transcript written on newtonsubdivision9641.log.
↦ This is dvips(k) 5.997 Copyright 2017 Radical Eye Software (www.radicaley\
  e.com)
↦ dvips: DVI file can’t be opened: /tmp/newtonsubdivision9641.dvi: No such \
  file or directory
↦ rm: das Entfernen von ’newtonsubdivision9641.aux’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’newtonsubdivision9641.ps?’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ rm: das Entfernen von ’newtonsubdivision9641.dvi’ ist nicht mglich: Dat\
  ei oder Verzeichnis nicht gefunden
↦ 0
↦ sh: xdg-open: Kommando nicht gefunden.
```

D.13.6.8 tropicalJInvariant

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicalJInvariant(f[,#]); f poly or list, # optional list

Assume: f is list of linear polynomials of the form ax+by+c with integers a, b and a rational number c representing a tropical Laurent polynomial defining a tropical plane curve; alternatively f can be a polynomial in Q(t)[x,y] defining a tropical plane curve via the valuation map;

the basering must have a global monomial ordering, two variables and up to one parameter!

Return: number, if the graph underlying the tropical curve has precisely one loop then its weighted lattice length is returned, otherwise the result will be -1

Note: - if the tropical curve is elliptic and its embedded graph has precisely one loop, then the weighted lattice length of the loop is its tropical j-invariant
- the procedure checks if the embedded graph of the tropical curve has genus one, but it does NOT check if the loop can be resolved, so that the curve is not a proper tropical elliptic curve
- if the embedded graph of a tropical elliptic curve has more than one loop, then all but one can be resolved, but this is not observed by this procedure, so it will not compute the j-invariant
- if # is empty, then the tropical curve is computed w.r.t. minimum, if #[1] is the string 'max', then it is computed w.r.t. maximum
- the tropicalJInvariant of a plane tropical cubic is the 'cycle length' of the cubic as introduced in the paper: Eric Katz, Hannah Markwig, Thomas Markwig: The j-invariant of a cubic tropical plane curve.

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
// tropcialJInvariant computes the tropical j-invariant of an elliptic curve
tropicalJInvariant(t*(x3+y3+1)+1/t*(x2+y2+x+y+x2y+xy2)+1/t2*xy);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 6
// the Newton polygon need not be the standard simplex
tropicalJInvariant(x+y+x2y+xy2+1/t*xy);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 8
// the curve can have arbitrary degree
tropicalJInvariant(t*(x7+y7+1)+1/t*(x4+y4+x2+y2+x3y+xy3)+1/t7*x2y2);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 14/3
// the procedure does not realise, if the embedded graph of the tropical
//     curve has a loop that can be resolved
tropicalJInvariant(1+x+y+xy+tx2y+txy2);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 4
// but it does realise, if the curve has no loop at all ...
tropicalJInvariant(x+y+1);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ The embedded graph of the curve has not genus one.
↦ -1
// or if the embedded graph has more than one loop - even if only one
//     cannot be resolved
tropicalJInvariant(1+x+y+xy+tx2y+txy2+t3x5+t3y5+tx2y2+t2xy4+t2yx4);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ The embedded graph of the curve has not genus one.
↦ -1
```

D.13.6.9 weierstrassForm

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

D.13.6.10 conicWithTangents

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: conicWithTangents(points[,#]); points list, # optional list

Assume: points is a list of five points in the plane over K(t)

Return: list, l[1] = the list points of the five given points
l[2] = the conic f passing through the five points
l[3] = list of equations of tangents to f in the given points
l[4] = ideal, tropicalisation of f (i.e. list of linear forms)
l[5] = a list of the tropicalisation of the tangents
l[6] = a list containing the vertices of the tropical conic f
l[7] = a list containing lists with vertices of the tangents
l[8] = a string which contains the latex-code to draw the tropical conic and its tropicalised tangents
l[9] = if # is non-empty, this is the same data for the dual conic and the points dual to the computed tangents

Note: the points must be generic, i.e. no three on a line

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
// the input consists of a list of five points in the plane over Q(t)
list points=list(1/t2,t),list(1/t,t2),list(1,1),list(t,1/t2),list(t2,1/t);
list conic=conicWithTangents(points);
// conic[1] is the list of the given five points
conic[1];
↦ [1]:
↦    [1]:
↦ 1/(t2)
↦    [2]:
↦ (t)
↦ [2]:
↦    [1]:
↦ 1/(t)
↦    [2]:
↦ (t2)
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦ [4]:
↦    [1]:
```

```
↦ (t)
↦    [2]:
↦ 1/(t2)
↦ [5]:
↦    [1]:
↦ (t2)
↦    [2]:
↦ 1/(t)
// conic[2] is the equation of the conic f passing through the five points
conic[2];
↦ (2t3)*x2+(-t6+t5+2t4+t3+2t2+t-1)*xy+(2t3)*y2+(-t5-2t4-3t3-2t2-t)*x+(-t5-2\
  t4-3t3-2t2-t)*y+(t6+t5+2t4+t3+2t2+t+1)
// conic[3] is a list containing the equations of the tangents
//          through the five points
conic[3];
↦ [1]:
↦    (-t7+t6+t5-t4-t3-t2+2t)*x+(-t7+t6-2t5+2t2+t-1)/(t2)*y+(t7+t5-t4+t3-t2-\
  1)/(t)
↦ [2]:
↦    (-t8+t7+2t6-2t3+t2-t)*x+(2t6-t5-t4-t3+t2+t-1)/(t)*y+(-t7-t5+t4-t3+t2+1\
  )
↦ [3]:
↦    (-t6+2t3-1)*x+(-t6+2t3-1)*y+(2t6-4t3+2)
↦ [4]:
↦    (-t7+t6-2t5+2t2+t-1)/(t2)*x+(-t7+t6+t5-t4-t3-t2+2t)*y+(t7+t5-t4+t3-t2-\
  1)/(t)
↦ [5]:
↦    (2t6-t5-t4-t3+t2+t-1)/(t)*x+(-t8+t7+2t6-2t3+t2-t)*y+(-t7-t5+t4-t3+t2+1\
  )
// conic[4] is an ideal representing the tropicalisation of the conic f
conic[4];
↦ [1]:
↦    2*x+3
↦ [2]:
↦    x+y
↦ [3]:
↦    2*y+3
↦ [4]:
↦    x+1
↦ [5]:
↦    y+1
↦ [6]:
↦    0
// conic[5] is a list containing the tropicalisation
//          of the five tangents in conic[3]
conic[5];
↦ [1]:
↦    [1]:
↦       x+1
↦    [2]:
↦       y-2
↦    [3]:
↦       -1
↦ [2]:
↦    [1]:
↦       x+1
↦    [2]:
↦       y-1
↦    [3]:
↦       0
↦ [3]:
↦    [1]:
↦       x
↦    [2]:
↦       y
↦    [3]:
↦       0
↦ [4]:
↦    [1]:
↦       x-2
↦    [2]:
↦       y+1
↦    [3]:
↦       -1
↦ [5]:
↦    [1]:
↦       x-1
↦    [2]:
↦       y+1
↦    [3]:
↦       0
// conic[6] is a list containing the vertices of the tropical conic
conic[6];
↦ [1]:
↦    [1]:
↦       -2
↦    [2]:
↦       1
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       -2
↦ [3]:
↦    [1]:
↦       -1
↦    [2]:
↦       1
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       -1
// conic[7] is a list containing the vertices of the five tangents
conic[7];
↦ [1]:
```

```
↦    [1]:
↦       [1]:
↦          -2
↦       [2]:
↦          1
↦ [2]:
↦    [1]:
↦       [1]:
↦          -1
↦       [2]:
↦          1
↦ [3]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦ [4]:
↦    [1]:
↦       [1]:
↦          1
↦       [2]:
↦          -2
↦ [5]:
↦    [1]:
↦       [1]:
↦          1
↦       [2]:
↦          -1
// conic[8] contains the latex code to draw the tropical conic and
//          its tropicalised tangents; it can written in a file, processed and
//          displayed via xdg-open
write(":w /tmp/conic.tex",conic[8]);
system("sh","cd /tmp; latex /tmp/conic.tex; dvips /tmp/conic.dvi -o;
xdg-open conic.ps &");
↦ This is pdfTeX, Version 3.14159265-2.6-1.40.18 (TeX Live 2017) (preloaded\
   format=latex)
↦  restricted \write18 enabled.
↦ entering extended mode
↦ (/tmp/conic.tex
↦ LaTeX2e <2017/01/01> patch level 3
↦ Babel <3.10> and hyphenation patterns for 3 language(s) loaded.
↦ (/usr/share/texmf-dist/tex/latex/amscls/amsart.cls
↦ Document Class: amsart 2015/03/04 v2.20.2
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsmath.sty
↦ For additional information on amsmath, use the ‘?’ option.
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amstext.sty
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsgen.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsbsy.sty)
↦ (/usr/share/texmf-dist/tex/latex/amsmath/amsopn.sty))
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/umsa.fd)
↦ (/usr/share/texmf-dist/tex/latex/amsfonts/amsfonts.sty))
↦
```

```
↦ ! LaTeX Error: File ‘texdraw.sty’ not found.
↦ 
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦ 
↦ Enter file name: warning: kpathsea: $.sty: Unrecognized variable construc\
   t ‘$.’.
↦ 
↦ ! LaTeX Error: File ‘$.sty’ not found.
↦ 
↦ Type X to quit or <RETURN> to proceed,
↦ or enter new name. (Default extension: sty)
↦ 
↦ Enter file name: 
↦ ! Emergency stop.
↦ <read *> 
↦ 
↦ l.3 \begin
↦           {document}^^M
↦ No pages of output.
↦ Transcript written on conic.log.
↦ This is dvips(k) 5.997 Copyright 2017 Radical Eye Software (www.radicaley\
   e.com)
↦ dvips: DVI file can’t be opened: /tmp/conic.dvi: No such file or director\
   y
↦ 0
↦ sh: Zeile 1: xdg-open: Kommando nicht gefunden.
// with an optional argument the same information for the dual conic is computed
//         and saved in conic[9]
conic=conicWithTangents(points,1);
conic[9][2]; // the equation of the dual conic
↦ (t6-6t5+11t4-6t3+t2)*x2+(-4t8+8t7-2t6-4t5+6t4-4t3-2t2+8t-4)*xy+(t6-6t5+11\
   t4-6t3+t2)*y2+(-2t7+2t6+4t5-6t4+4t3+2t2-2t)*x+(-2t7+2t6+4t5-6t4+4t3+2t2-2\
   t)*y+(t8-4t7+2t6+8t5-13t4+8t3+2t2-4t+1)
```

D.13.6.11 tropicalise

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicalise(f[,#]); f polynomial, # optional list

Assume: f is a polynomial in Q(t)[x_1,...,x_n]

Return: list, the linear forms of the tropicalisation of f

Note: if # is empty, then the valuation of t will be 1,
if # is the string 'max' it will be -1;
the latter supposes that we consider the maximum of the computed linear forms, the former that we consider their minimum

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
tropicalise(2t3x2-1/t*xy+2t3y2+(3t3-t)*x+ty+(t6+1));
↦ [1]:
```

```
↦    2*x+3
↦ [2]:
↦    x+y-1
↦ [3]:
↦    2*y+3
↦ [4]:
↦    x+1
↦ [5]:
↦    y+1
↦ [6]:
↦    0
```

D.13.6.12 tropicaliseSet

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicaliseSet(i); i ideal

Assume: i is an ideal in Q(t)[x_1,...,x_n]

Return: list, the jth entry is the tropicalisation of the jth generator of i

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
ideal i=txy-y2+1,2t3x2+1/t*y-t6;
tropicaliseSet(i);
↦ [1]:
↦    [1]:
↦       x+y+1
↦    [2]:
↦       2*y
↦    [3]:
↦       0
↦ [2]:
↦    [1]:
↦       2*x+3
↦    [2]:
↦       y-1
↦    [3]:
↦       6
```

D.13.6.13 tInitialForm

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tInitialForm(f,w); f a polynomial, w an integer vector

Assume: f is a polynomial in Q[t,x_1,...,x_n] and w=(w_0,w_1,...,w_n)

Return: poly, the t-initialform of f(t,x) w.r.t. w evaluated at t=1

Note: the t-initialform is the sum of the terms with MAXIMAL
weighted order w.r.t. w

Example:

```
LIB "tropical.lib";
ring r=0,(t,x,y),dp;
poly f=t4x2+y2-t2xy+t4x-t9;
intvec w=-1,-2,-3;
tInitialForm(f,w);
↦ y2+x
```

D.13.6.14 tInitialIdeal

Procedure from library tropical.lib (see Section D.13.6 [tropical_lib], page 2139).

Usage: tInitialIdeal(i,w); i ideal, w intvec

Assume: i is an ideal in Q[t,x_1,...,x_n] and w=(w_0,...,w_n)

Return: ideal ini, the t-initial ideal of i with respect to w

Example:

```
LIB "tropical.lib";
ring r=0,(t,x,y),dp;
ideal i=t2x-y+t3,t2x-y-2t3x;
intvec w=-1,2,0;
// the t-initial forms of the generators are
tInitialForm(i[1],w),tInitialForm(i[2],w);
↦ x-y x-y
// and they do not generate the t-initial ideal of i
tInitialIdeal(i,w);
↦ _[1]=-x+y
↦ _[2]=2xy
↦ _[3]=2x
↦ _[4]=2xy2
```

D.13.6.15 initialForm

Procedure from library tropical.lib (see Section D.13.6 [tropical_lib], page 2139).

Usage: initialForm(f,w); f a polynomial, w an integer vector

Assume: f is a polynomial in Q[x_1,...,x_n] and w=(w_1,...,w_n)

Return: poly, the initial form of f(x) w.r.t. w

Note: the initialForm consists of the terms with MAXIMAL weighted order w.r.t. w

Example:

```
LIB "tropical.lib";
ring r=0,(x,y),dp;
poly f=x3+y2-xy+x-1;
intvec w=2,3;
initialForm(f,w);
↦ x3+y2
```

D.13.6.16 initialIdeal

Procedure from library tropical.lib (see Section D.13.6 [tropical_lib], page 2139).

Usage: initialIdeal(i,w); i ideal, w intvec

Assume: i is an ideal in Q[x_1,...,x_n] and w=(w_1,...,w_n)

Return: ideal, the initialIdeal of i w.r.t. w

Note: the initialIdeal consists of the terms with MAXIMAL weighted order w.r.t. w

Example:

```
LIB "tropical.lib";
ring r=0,(x,y),dp;
poly f=x3+y2-xy+x-1;
intvec w=2,3;
initialIdeal(f,w);
↦ _[1]=x3+y2
```

D.13.6.17 texNumber

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texNumber(f); f poly

Return: string, tex command representing leading coefficient of f using \frac

Example:

```
LIB "tropical.lib";
ring r=(0,t),x,dp;
texNumber((3t2-1)/t3);
↦ \tfrac{-1+3\cdot t^{2}}{t^{3}}
```

D.13.6.18 texPolynomial

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texPolynomial(f); f poly

Return: string, the tex command representing f

Example:

```
LIB "tropical.lib";
ring r=(0,t),x,dp;
texPolynomial(1/t*x2-t2x+1/t);
↦ \tfrac{1}{t}\cdot x^{2}-t^{2}\cdot x+\tfrac{1}{t}
```

D.13.6.19 texMatrix

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texMatrix(M); M matrix

Return: string, the tex command representing M

Example:

```
LIB "tropical.lib";
ring r=(0,t),x,dp;
matrix M[2][2]=3/2,1/t*x2-t2x+1/t,5,-2x;
texMatrix(M);
↦ \left(\begin{array}{cc}
↦    \tfrac{3}{2} & \tfrac{1}{t}\cdot x^{2}-t^{2}\cdot x+\tfrac{1}{t} \\
↦      5 & -2\cdot x \\
↦      \end{array}\right)
```

D.13.6.20 texDrawBasic

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texDrawBasic(texdraw); list texdraw

Assume: texdraw is a list of strings representing texdraw commands (as produced by texDrawTropical) which should be embedded into a texdraw environment

Return: string, a texdraw environment enclosing the input

Note: is called from conicWithTangents

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=x+y+1;
string texf=texDrawTropical(tropicalCurve(f),list("",1));
texDrawBasic(texf);
↦
↦      \begin{texdraw}
↦         \drawdim cm  \relunitscale 0.7 \arrowheadtype t:V
↦         %\linewd 0.05 \lpatt (0.1 0.4)
↦         %\move (-4 0) \avec (9 0) \move (0 -4) \avec (0 9)
↦         \linewd 0.1  \lpatt (1 0)
↦
↦
↦         \setgray 0.6
↦
↦
↦         \relunitscale 12
↦         \move (0 0) \fcir f:0 r:0.016
↦         \move (0 0) \rlvec (0 0.25)
↦         \move (0 0) \rlvec (0.25 0)
↦         \move (0 0) \rlvec (-0.25 -0.25)
↦
↦     %% HERE STARTS THE CODE FOR THE LATTICE
↦          \move (0 0) \fcir f:0.8 r:0.008
↦     %% HERE ENDS THE CODE FOR THE LATTICE
↦
↦      \end{texdraw}
```

D.13.6.21 texDrawTropical

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texDrawTropical(graph[,#]); graph list, # optional list

Assume: graph is the output of tropicalCurve

Return: string, the texdraw code of the tropical plane curve encoded by graph

Note: - if the list # is non-empty, the first entry should be a string; if this string is 'max', then the tropical curve is considered with respect to the maximum
- the procedure computes a scalefactor for the texdraw command which should help to display the curve in the right way; this may, however, be a bad idea if several texDrawTropical outputs are put together to form one image; the scalefactor can be

prescribed by the further optional entry of type poly
- one can add a string as last opional argument to the list #; it can be used to insert further texdraw commands (e.g. to have a lighter image as when called from inside conicWithTangents);
- the list # is optional and may as well be empty

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=x+y+x2y+xy2+1/t*xy;
list graph=tropicalCurve(f);
// compute the texdraw code of the tropical curve defined by f
texDrawTropical(graph);
↦
↦
↦         \setgray 0.6
↦
↦
↦         \relunitscale 6
↦         \move (-1 -1) \fcir f:0 r:0.03
↦         \move (-1 -1) \lvec (-1 1)
↦         \move (-1 -1) \lvec (1 -1)
↦         \move (-1 -1) \rlvec (-0.5 -0.5)
↦         \move (-1 1) \fcir f:0 r:0.03
↦         \move (-1 1) \lvec (1 1)
↦         \move (-1 1) \rlvec (-0.5 0.5)
↦         \move (1 -1) \fcir f:0 r:0.03
↦         \move (1 -1) \lvec (1 1)
↦         \move (1 -1) \rlvec (0.5 -0.5)
↦         \move (1 1) \fcir f:0 r:0.03
↦         \move (1 1) \rlvec (0.5 0.5)
↦
↦     %% HERE STARTS THE CODE FOR THE LATTICE
↦          \move (-1 -1) \fcir f:0.8 r:0.01
↦          \move (-1 0) \fcir f:0.8 r:0.01
↦          \move (-1 1) \fcir f:0.8 r:0.01
↦          \move (0 -1) \fcir f:0.8 r:0.01
↦          \move (0 0) \fcir f:0.8 r:0.01
↦          \move (0 1) \fcir f:0.8 r:0.01
↦          \move (1 -1) \fcir f:0.8 r:0.01
↦          \move (1 0) \fcir f:0.8 r:0.01
↦          \move (1 1) \fcir f:0.8 r:0.01
↦     %% HERE ENDS THE CODE FOR THE LATTICE
↦
// compute the texdraw code again, but set the scalefactor to 1
texDrawTropical(graph,"",1);
↦
↦
↦         \setgray 0.6
↦
↦
↦         \relunitscale 6
↦         \move (-1 -1) \fcir f:0 r:0.03
↦          \move (-1 -1) \lvec (-1 1)
↦          \move (-1 -1) \lvec (1 -1)
↦          \move (-1 -1) \rlvec (-0.5 -0.5)
↦          \move (-1 1) \fcir f:0 r:0.03
↦          \move (-1 1) \lvec (1 1)
↦          \move (-1 1) \rlvec (-0.5 0.5)
↦          \move (1 -1) \fcir f:0 r:0.03
↦          \move (1 -1) \lvec (1 1)
↦          \move (1 -1) \rlvec (0.5 -0.5)
↦          \move (1 1) \fcir f:0 r:0.03
↦          \move (1 1) \rlvec (0.5 0.5)
↦
↦      %% HERE STARTS THE CODE FOR THE LATTICE
↦           \move (-1 -1) \fcir f:0.8 r:0.01
↦           \move (-1 0) \fcir f:0.8 r:0.01
↦           \move (-1 1) \fcir f:0.8 r:0.01
↦           \move (0 -1) \fcir f:0.8 r:0.01
↦           \move (0 0) \fcir f:0.8 r:0.01
↦           \move (0 1) \fcir f:0.8 r:0.01
↦           \move (1 -1) \fcir f:0.8 r:0.01
↦           \move (1 0) \fcir f:0.8 r:0.01
↦           \move (1 1) \fcir f:0.8 r:0.01
↦      %% HERE ENDS THE CODE FOR THE LATTICE
↦
```

D.13.6.22 texDrawNewtonSubdivision

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texDrawNewtonSubdivision(graph[,#]); graph list, # optional list

Assume: graph is the output of tropicalCurve

Return: string, the texdraw code of the Newton subdivision of the tropical plane curve encoded by graph

Note: - the list # may contain optional arguments, of which only one will be considered, namely the first entry of type 'poly'; this entry should be a rational number which specifies the scaling factor to be used; if it is missing, the factor will be computed; the list # may as well be empty
- note that lattice points in the Newton subdivision which are black correspond to markings of the marked subdivision, while lattice points in grey are not marked

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=x+y+x2y+xy2+1/t*xy;
list graph=tropicalCurve(f);
// compute the texdraw code of the Newton subdivision of the tropical curve
texDrawNewtonSubdivision(graph);
↦
↦       \begin{texdraw}
↦          \drawdim cm  \relunitscale 1
↦          \linewd 0.05
↦           \move (1 2)
↦          \lvec (2 1)
↦          \move (2 1)
↦          \lvec (1 0)
↦          \move (1 0)
↦          \lvec (0 1)
↦          \move (0 1)
↦          \lvec (1 2)
↦
↦
↦          \move (2 1)
↦          \lvec (1 1)
↦          \move (1 1)
↦          \lvec (1 2)
↦          \move (1 0)
↦          \lvec (1 1)
↦          \move (1 1)
↦          \lvec (0 1)
↦          \move (0 0) \fcir f:0.6 r:0.03
↦          \move (0 1) \fcir f:0.6 r:0.03
↦          \move (0 2) \fcir f:0.6 r:0.03
↦          \move (1 0) \fcir f:0.6 r:0.03
↦          \move (1 1) \fcir f:0.6 r:0.03
↦          \move (1 2) \fcir f:0.6 r:0.03
↦          \move (2 0) \fcir f:0.6 r:0.03
↦          \move (2 1) \fcir f:0.6 r:0.03
↦          \move (2 2) \fcir f:0.6 r:0.03
↦         \move (2 1)
↦         \fcir f:0 r:0.04
↦         \move (1 2)
↦         \fcir f:0 r:0.04
↦         \move (1 1)
↦         \fcir f:0 r:0.04
↦         \move (1 0)
↦         \fcir f:0 r:0.04
↦         \move (0 1)
↦         \fcir f:0 r:0.04
↦     \end{texdraw}
↦
```

D.13.6.23 texDrawTriangulation

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: texDrawTriangulation(triang,polygon); triang,polygon list

Assume: polygon is a list of integer vectors describing the
lattice points of a marked polygon;
triang is a list of integer vectors describing a
triangulation of the marked polygon
in the sense that an integer vector of the form (i,j,k) describes the triangle formed by polygon[i], polygon[j] and polygon[k]

Return: string, a texdraw code for the triangulation described by triang without the texdraw environment

Example:

```
LIB "tropical.lib";
// the lattice polygon spanned by the points (0,0), (3,0) and (0,3)
// with all integer points as markings
list polygon=intvec(1,1),intvec(3,0),intvec(2,0),intvec(1,0),intvec(0,0),
intvec(2,1),intvec(0,1),intvec(1,2),intvec(0,2),intvec(0,3);
// define a triangulation by connecting the only interior point
//        with the vertices
list triang=intvec(1,2,5),intvec(1,5,10),intvec(1,2,10);
// produce the texdraw output of the triangulation triang
texDrawTriangulation(triang,polygon);
↦
↦         \drawdim cm  \relunitscale 1.2 \arrowheadtype t:V
↦
↦         \setgray 0
↦         \move (1 1)
↦         \fcir f:0 r:0.08
↦         \move (3 0)
↦         \fcir f:0 r:0.08
↦         \move (0 0)
↦         \fcir f:0 r:0.08
↦         \move (0 3)
↦         \fcir f:0 r:0.08
↦         \move (1 1)
↦         \lvec (3 0)
↦         \move (3 0)
↦         \lvec (0 0)
↦         \move (0 0)
↦         \lvec (1 1)
↦         \move (0 0)
↦         \lvec (0 3)
↦         \move (0 3)
↦         \lvec (1 1)
↦         \move (3 0)
↦         \lvec (0 3)
↦         \move (1 1)
↦         \fcir f:0.7 r:0.04
↦         \move (3 0)
↦         \fcir f:0.7 r:0.04
↦         \move (2 0)
↦         \fcir f:0.7 r:0.04
↦         \move (1 0)
↦         \fcir f:0.7 r:0.04
↦         \move (0 0)
↦         \fcir f:0.7 r:0.04
↦         \move (2 1)
↦         \fcir f:0.7 r:0.04
↦         \move (0 1)
↦         \fcir f:0.7 r:0.04
↦         \move (1 2)
↦         \fcir f:0.7 r:0.04
↦         \move (0 2)
↦         \fcir f:0.7 r:0.04
↦          \move (0 3)
↦          \fcir f:0.7 r:0.04
```

D.13.6.24 radicalMemberShip

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: radicalMemberShip (f,i); f poly, i ideal

Return: int, 1 if f is in the radical of i, 0 else

Example:

```
LIB "tropical.lib";
ring r=0,(x,y),dp;
ideal i=(x+1)*y2;
// y is NOT in the radical of i
radicalMemberShip(y,i);
↦ 0
ring rr=0,(x,y),ds;
ideal i=(x+1)*y2;
// since this time the ordering is local, y is in the radical of i
radicalMemberShip(y,i);
↦ 1
```

D.13.6.25 tInitialFormPar

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tInitialFormPar(f,w); f a polynomial, w an integer vector

Assume: f is a polynomial in Q(t)[x_1,...,x_n] and w=(w_1,...,w_2)

Return: poly, the t-initialform of f(t,x) w.r.t. (1,w) evaluated at t=1

Note: the t-initialform are the terms with MINIMAL weighted order w.r.t. (1,w)

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t4x2+y2-t2xy+t4x-t9;
intvec w=2,3;
tInitialFormPar(f,w);
↦ y2+x
```

D.13.6.26 tInitialFormParMax

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tInitialFormParMax(f,w); f a polynomial, w an integer vector

Assume: f is a polynomial in Q(t)[x_1,...,x_n] and w=(w_1,...,w_2)

Return: poly, the t-initialform of f(t,x) w.r.t. (-1,w) evaluated at t=1

Note: the t-initialform are the terms with MAXIMAL weighted order w.r.t. (1,w)

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t4x2+y2-t2xy+t4x-1/t6;
intvec w=2,3;
tInitialFormParMax(f,w);
↦ y2-1
```

D.13.6.27 solveTInitialFormPar

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: solveTInitialFormPar(i); i ideal

Assume: i is a zero-dimensional ideal in Q(t)[x_1,...,x_n] generated by the (1,w)-homogeneous elements for some integer vector w - i.e. by the (1,w)-initialforms of polynomials

Return: none

Note: the procedure just displays complex approximations
of the solution set of i

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
ideal i=t2x2+y2,x-t2;
solveTInitialFormPar(i);
↦ [1]:
↦    [1]:
↦ 1
↦    [2]:
↦ -i
↦ [2]:
↦    [1]:
↦ 1
↦    [2]:
↦ i
↦ [1]:
↦    // coefficients: real[i](complex:8 digits, additional 8 digits)/(i^2+1\
)
↦ // number of vars : 2
↦ //        block   1 : ordering lp
↦ //                  : names    x y
↦ //        block   2 : ordering C
```

D.13.6.28 detropicalise

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: detropicalise(f); f poly or f list

Assume: if f is of type poly then t is a linear polynomial with an arbitrary constant term and positive integer coefficients as further coefficients;
if f is of type list then f is a list of polynomials of the type just described in before

Return: poly, the detropicalisation of f ignoring the constant parts

Note: the output will be a monomial and the constant coefficient has been ignored

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
detropicalise(3x+4y-1);
↦ x3y4
```

D.13.6.29 tDetropicalise

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tDetropicalise(f); f poly or f list

Assume: if f is of type poly then f is a linear polynomial with an integer constant term and positive integer coefficients as further coefficients;
if f is of type list then it is a list of polynomials of the type just described in before

Return: poly, the detropicalisation of f over the field Q(t)

Note: the output will be a term where the coeffiecient is a Laurent monomial in the variable t

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
tDetropicalise(3x+4y-1);
↦ 1/(t)*x3y4
```

D.13.6.30 dualConic

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: dualConic(f); f poly

Assume: f is an affine conic in two variables x and y

Return: poly, the equation of the dual conic

Example:

```
LIB "tropical.lib";
ring r=0,(x,y),dp;
poly conic=2x2+1/2y2-1;
dualConic(conic);
↦ 1/2x2+2y2-1
```

D.13.6.31 parameterSubstitute

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: parameterSubstitute(f,N); f poly, N int

Assume: f is a polynomial in Q(t)[x_1,...,x_n] describing
a plane curve over Q(t)

Return: poly f with t replaced by t^N

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t2xy+1/t*y+t3;
parameterSubstitute(f,3);
↦ (t6)*xy+1/(t3)*y+(t9)
parameterSubstitute(f,-1);
↦ 1/(t2)*xy+(t)*y+1/(t3)
```

D.13.6.32 tropicalSubst

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: parameterSubstitute(f,N,L); f poly, N int, L list

Assume: f is a polynomial in Q(t)[x_1,...,x_k]
and L is a list of the form var(i_1),poly_1,...,v(i_k),poly_k

Return: list, the list is the tropical polynomial which we get from f by replacing the i-th variable be the i-th polynomial but in the i-th polynomial the parameter t is replaced by t^1/N

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
poly f=t2x+1/t*y-1;
tropicalSubst(f,2,x,x+t,y,tx+y+t2);
↦ [1]:
↦    x-1/2
↦ [2]:
↦    y-1
↦ [3]:
↦    5/2
// The procedure can be used to study the effect of a transformation of
// the form x -> x+t^b, with b a rational number, on the tropicalisation and
// the j-invariant of a cubic over the Puiseux series.
f=t7*y3+t3*y2+t*(x3+xy2+y+1)+xy;
// - b=3/2, then the cycle length of the tropical cubic equals -val(j-inv)
list g32=tropicalSubst(f,2,x,x+t3,y,y);
tropicalJInvariant(g32);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 5
// - b=1, then it is still true, but only just ...
list g1=tropicalSubst(f,1,x,x+t,y,y);
tropicalJInvariant(g1);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 5
// - b=2/3, as soon as b<1, the cycle length is strictly less than -val(j-inv)
list g23=tropicalSubst(f,3,x,x+t2,y,y);
tropicalJInvariant(g23);
↦ // ** int division with ‘/‘: use ‘div‘ instead in line >>  genus=-genus/2\
   ; // we have counted each bounded edge twice<<
↦ 14/3
```

D.13.6.33 randomPolyInT

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: randomPolyInT(d,ug,og[,#]); d, ug, og int, # list

Assume: the basering has a parameter t

Return: poly, a polynomial of degree d where the coefficients are of the form t^j with j a random integer between ug and og

Note: if an optional argument # is given, then the coefficients are instead either of the form t^j as above or they are zero, and this is chosen randomly

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
randomPolyInT(3,-2,5);
↦ (t5)*x3+1/(t2)*x2y+(t3)*xy2+(t4)*y3+1/(t)*x2+(t5)*xy+y2+1/(t2)*x+1/(t)*y+\
   1
randomPolyInT(3,-2,5,1);
↦ (t4)*y3+1/(t)*x2+(t2)*xy+(t2)*y2+(t3)*x
```

D.13.6.34 cleanTmp

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: cleanTmp()

Purpose: some procedures create latex and ps-files in the directory /tmp; in order to remove them simply call cleanTmp();

Return: none

D.13.6.35 groebnerCone

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: groebnerCone(I,w); I ideal or poly, w intvec or bigintmat

Assume: I a reduced standard basis and w contained in the maximal Groebner cone

Return: cone, the Groebner cone of I with respect to w

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z),dp;
ideal I = cyclic(3);
option(redSB);
ideal stdI = std(I);
// w lies in the interior of a maximal Groebner cone
intvec w = 3,2,1;
cone CwI = groebnerCone(stdI,w);
print(rays(CwI));
↦ 1,1,0,
↦ 1,0,0,
↦ 1,1,1
// v lies on a facet of a maximal Groebner cone
```

```
intvec v = 2,1,0;
cone CvI = groebnerCone(stdI,v);
print(rays(CvI));
↦ 1,0,0,
↦ 1,1,0
// v lies on a ray of a maximal Groebner cone
intvec u = 1,1,1;
cone CuI = groebnerCone(stdI,u);
print(rays(CuI));
↦ 1,1,1
```

D.13.6.36 maximalGroebnerCone

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: maximalGroebnerCone(I); I ideal or poly

Assume: I a reduced standard basis

Return: cone, the maximal Groebner cone of I with respect to the current ordering

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z),dp;
ideal I = cyclic(3);
option(redSB);
ideal stdI = std(I);
cone CI = maximalGroebnerCone(stdI);
print(rays(CI));
↦ 1,1,0,
↦ 1,0,0,
↦ 1,1,1
ring s = 0,(x,y,z,u),dp;
ideal Ih = homog(cyclic(3),u);
ideal stdI = std(Ih);
cone CIh = maximalGroebnerCone(stdI);
print(rays(CIh));
↦ 1, 1,-1,-1,
↦ 3,-1,-1,-1,
↦ 1, 1, 1,-3
print(generatorsOfLinealitySpace(CIh));
↦ -1,-1,-1,-1
ring rw = 0,(x,y,z),(a(1,0,1),lp);
ideal I = cyclic(3);
ideal stdI = std(I);
CI = maximalGroebnerCone(stdI);
print(rays(CI));
↦ 1,0,0,
↦ 1,0,1,
↦ 1,1,1
```

D.13.6.37 homogeneitySpace

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: homogeneitySpace(I); I ideal or poly

Assume: I a reduced standard basis

Return: cone, the set of all weight vectors with respect to whom I is weighted homogeneous

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z),dp;
ideal I = cyclic(3);
option(redSB);
ideal stdI = std(I);
cone C0I = homogeneitySpace(stdI);
print(generatorsOfLinealitySpace(C0I));
↦
ring s = 0,(x,y,z,u),dp;
ideal Ih = homog(cyclic(3),u);
ideal stdI = std(Ih);
cone C0Ih = homogeneitySpace(stdI);
print(generatorsOfLinealitySpace(C0Ih));
↦ -1,-1,-1,-1
```

D.13.6.38 initial

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: initial(f,w); f poly, w intvec or bigintmat
initial(I,w); I ideal, w intvec or bigintmat

Assume: I reduced Groebner basis,
w in the maximal Groebner cone of I with respect to the current ordering

Return: poly or ideal, the initial form of f or the initial ideal of I with respect to w

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z),dp;
ideal I = cyclic(3);
intvec w = 1,1,1;
option(redSB);
ideal stdI = std(I);
stdI;
↦ stdI[1]=x+y+z
↦ stdI[2]=y2+yz+z2
↦ stdI[3]=z3-1
ideal inI = initial(stdI,w);
inI;
↦ inI[1]=x+y+z
↦ inI[2]=y2+yz+z2
↦ inI[3]=z3
```

D.13.6.39 tropicalVariety

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: tropicalVariety(f[,p]); f poly, p optional number
tropicalVariety(I[,p]); I ideal, p optional number

Assume: I homogeneous, p prime number

Return: fan, the tropical variety of f resp. I with respect to the trivial valuation or the p-adic valuation

Note: set printlevel=1 for output during traversal

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z,w),dp;
ideal I = x-2y+3z,3y-4z+5w;
tropicalVariety(I);
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 4
↦
↦ DIM
↦ 2
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -3 1 1 1 # 0
↦ 1 -3 1 1 # 1
↦ 1 1 -3 1 # 2
↦ 1 1 1 -3 # 3
↦
↦ N_RAYS
↦ 4
↦
↦ LINEALITY_SPACE
↦ -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ 1 -1 0 0 # 0
↦ 1 0 -1 0 # 1
↦ 1 0 0 -1 # 2
↦
↦ F_VECTOR
↦ 1 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦
↦ MAXIMAL_CONES
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦
tropicalVariety(I,number(2));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 3
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -2 -1 1 -1 1 # 0
↦ -1 1 -1 1 -1 # 1
↦ 0 -3 1 1 1 # 2
↦ 0 1 -3 1 1 # 3
↦ 0 1 1 -3 1 # 4
↦ 0 1 1 1 -3 # 5
↦
↦ N_RAYS
↦ 6
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 6 5
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 4}
↦ {1 3}
↦ {1 5}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 4}
↦ {1 3}
↦ {1 5}
↦
tropicalVariety(I,number(3));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 3
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -2 -1 -1 1 1 # 0
↦ -2 1 1 -1 -1 # 1
↦ 0 -3 1 1 1 # 2
↦ 0 1 -3 1 1 # 3
↦ 0 1 1 -3 1 # 4
↦ 0 1 1 1 -3 # 5
↦
↦ N_RAYS
↦ 6
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
```

```
↦ F_VECTOR
↦ 1 6 5
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {1 4}
↦ {1 5}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {1 4}
↦ {1 5}
↦
tropicalVariety(I,number(5));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 3
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -4 -1 -1 -1 3 # 0
↦ 0 -3 1 1 1 # 1
↦ 0 1 -3 1 1 # 2
↦ 0 1 1 -3 1 # 3
↦ 0 1 1 1 -3 # 4
↦
↦ N_RAYS
↦ 5
```

```
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 5 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦
tropicalVariety(I,number(7));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 3
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -1 0 0 0 0 # 0
↦ 0 -3 1 1 1 # 1
↦ 0 1 -3 1 1 # 2
↦ 0 1 1 -3 1 # 3
↦ 0 1 1 1 -3 # 4
↦
↦ N_RAYS
↦ 5
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 5 4
↦
↦ SIMPLICIAL
↦ 1
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦
↦ MAXIMAL_CONES
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦
```

D.13.6.40 groebnerFan

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: groebnerFan(f); f poly
groebnerFan(I); I ideal

Assume: I homogeneous

Return: fan, the Groebner fan of f resp. I

Note: set printlevel=1 for output during traversal

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z,w),dp;
ideal I = x-2y+3z,3y-4z+5w;
groebnerFan(I);
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 4
↦
↦ DIM
↦ 4
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -3 1 1 1 # 0
↦ -1 -1 -1 3 # 1
↦ -1 -1 3 -1 # 2
↦ -1 3 -1 -1 # 3
↦ 1 -3 1 1 # 4
↦ 1 1 -3 1 # 5
↦ 1 1 1 -3 # 6
↦ 3 -1 -1 -1 # 7
↦
↦ N_RAYS
↦ 8
↦
↦ LINEALITY_SPACE
↦ -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ 1 -1 0 0 # 0
↦ 1 0 -1 0 # 1
↦ 1 0 0 -1 # 2
↦
↦ F_VECTOR
↦ 1 8 12 6
↦
↦ SIMPLICIAL
↦ 0
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {6}
↦ {7}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {1 4}
↦ {2 4}
↦ {1 5}
↦ {2 6}
↦ {3 5}
↦ {3 6}
↦ {4 7}
↦ {5 7}
↦ {6 7}
↦ {0 1 2 4} # Dimension 4
↦ {0 1 3 5}
↦ {0 2 3 6}
↦ {1 4 5 7}
↦ {2 4 6 7}
↦ {3 5 6 7}
↦
↦ MAXIMAL_CONES
↦ {0 1 2 4} # Dimension 4
↦ {0 1 3 5}
↦ {0 2 3 6}
↦ {1 4 5 7}
↦ {2 4 6 7}
↦ {3 5 6 7}
↦
```

D.13.6.41 groebnerComplex

Procedure from library `tropical.lib` (see Section D.13.6 [tropical_lib], page 2139).

Usage: groebnerComplex(f,p); f poly, p number
groebnerComplex(I,p); I ideal, p number

Assume: I homogeneous, p prime number

Return: fan, the Groebner complex of f resp. I with respect to the p-adic valuation

Note: set printlevel=1 for output during traversal

Example:

```
LIB "tropical.lib";
ring r = 0,(x,y,z,w),dp;
ideal I = x-2y+3z,3y-4z+5w;
groebnerComplex(I,number(2));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 5
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -2 -1 1 -1 1 # 0
↦ -1 1 -1 1 -1 # 1
↦ 0 -3 1 1 1 # 2
↦ 0 -1 -1 -1 3 # 3
↦ 0 -1 -1 3 -1 # 4
↦ 0 -1 3 -1 -1 # 5
↦ 0 1 -3 1 1 # 6
↦ 0 1 1 -3 1 # 7
↦ 0 1 1 1 -3 # 8
↦ 0 3 -1 -1 -1 # 9
↦
↦ N_RAYS
↦ 10
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 10 23 22 8
↦
↦ SIMPLICIAL
↦ 0
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {6}
↦ {7}
↦ {8}
↦ {9}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 5}
↦ {0 7}
↦ {1 4}
↦ {1 6}
↦ {1 8}
↦ {1 9}
↦ {2 3}
↦ {2 4}
↦ {2 5}
↦ {2 7}
↦ {3 6}
↦ {4 6}
↦ {3 7}
↦ {4 8}
↦ {5 7}
↦ {5 8}
↦ {6 8}
↦ {6 9}
↦ {7 9}
↦ {8 9}
↦ {0 1 2 4} # Dimension 4
↦ {0 1 3 6}
↦ {0 1 5 8}
↦ {0 1 7 9}
↦ {0 2 3}
↦ {0 2 5}
↦ {0 2 7}
↦ {0 3 7}
↦ {0 5 7}
↦ {1 4 6}
↦ {1 4 8}
↦ {1 6 8}
↦ {1 6 9}
↦ {1 8 9}
↦ {2 3 4 6}
↦ {2 4 5 8}
↦ {2 3 7}
↦ {2 5 7}
↦ {4 6 8}
↦ {3 6 7 9}
↦ {5 7 8 9}
↦ {6 8 9}
↦ {0 1 2 3 4 6} # Dimension 5
↦ {0 1 2 4 5 8}
↦ {0 1 3 6 7 9}
↦ {0 1 5 7 8 9}
↦ {0 2 3 7}
↦ {0 2 5 7}
↦ {1 4 6 8}
↦ {1 6 8 9}
↦
↦ MAXIMAL_CONES
↦ {0 1 2 3 4 6} # Dimension 5
↦ {0 1 2 4 5 8}
↦ {0 1 3 6 7 9}
↦ {0 1 5 7 8 9}
↦ {0 2 3 7}
↦ {0 2 5 7}
↦ {1 4 6 8}
↦ {1 6 8 9}
↦
groebnerComplex(I,number(3));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 5
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -2 -1 -1 1 1 # 0
↦ -2 1 1 -1 -1 # 1
↦ 0 -3 1 1 1 # 2
↦ 0 -1 -1 -1 3 # 3
↦ 0 -1 -1 3 -1 # 4
↦ 0 -1 3 -1 -1 # 5
↦ 0 1 -3 1 1 # 6
↦ 0 1 1 -3 1 # 7
↦ 0 1 1 1 -3 # 8
↦ 0 3 -1 -1 -1 # 9
↦
↦ N_RAYS
↦ 10
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 10 23 22 8
↦
↦ SIMPLICIAL
↦ 0
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {6}
↦ {7}
↦ {8}
↦ {9}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦ {0 6}
↦ {1 5}
↦ {1 7}
↦ {1 8}
↦ {1 9}
↦ {2 3}
↦ {2 4}
↦ {2 5}
↦ {2 6}
↦ {3 6}
↦ {4 6}
↦ {3 7}
↦ {4 8}
↦ {5 7}
↦ {5 8}
↦ {7 8}
↦ {6 9}
↦ {7 9}
↦ {8 9}
↦ {0 1 2 5} # Dimension 4
↦ {0 1 3 7}
↦ {0 1 4 8}
↦ {0 1 6 9}
↦ {0 2 3}
↦ {0 2 4}
↦ {0 2 6}
↦ {0 3 6}
↦ {0 4 6}
↦ {1 5 7}
↦ {1 5 8}
↦ {1 7 8}
↦ {1 7 9}
↦ {1 8 9}
↦ {2 3 5 7}
↦ {2 4 5 8}
↦ {2 3 6}
↦ {2 4 6}
↦ {5 7 8}
↦ {3 6 7 9}
↦ {4 6 8 9}
↦ {7 8 9}
↦ {0 1 2 3 5 7} # Dimension 5
↦ {0 1 2 4 5 8}
↦ {0 1 3 6 7 9}
↦ {0 1 4 6 8 9}
↦ {0 2 3 6}
↦ {0 2 4 6}
↦ {1 5 7 8}
↦ {1 7 8 9}
↦
↦ MAXIMAL_CONES
↦ {0 1 2 3 5 7} # Dimension 5
↦ {0 1 2 4 5 8}
↦ {0 1 3 6 7 9}
↦ {0 1 4 6 8 9}
↦ {0 2 3 6}
↦ {0 2 4 6}
↦ {1 5 7 8}
↦ {1 7 8 9}
↦
groebnerComplex(I,number(5));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 5
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -4 -1 -1 -1 3 # 0
↦ 0 -3 1 1 1 # 1
↦ 0 -1 -1 -1 3 # 2
↦ 0 -1 -1 3 -1 # 3
↦ 0 -1 3 -1 -1 # 4
↦ 0 1 -3 1 1 # 5
↦ 0 1 1 -3 1 # 6
↦ 0 1 1 1 -3 # 7
↦ 0 3 -1 -1 -1 # 8
↦
↦ N_RAYS
↦ 9
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 9 20 18 6
↦
↦ SIMPLICIAL
↦ 0
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {6}
↦ {7}
↦ {8}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦ {0 5}
↦ {0 6}
↦ {0 7}
↦ {0 8}
↦ {1 2}
↦ {1 3}
↦ {1 4}
↦ {2 5}
↦ {3 5}
↦ {2 6}
↦ {3 7}
↦ {4 6}
↦ {4 7}
↦ {5 8}
↦ {6 8}
↦ {7 8}
↦ {0 1 2} # Dimension 4
↦ {0 1 3}
↦ {0 1 4}
↦ {0 2 5}
↦ {0 3 5}
↦ {0 2 6}
↦ {0 3 7}
↦ {0 4 6}
↦ {0 4 7}
↦ {0 5 8}
↦ {0 6 8}
↦ {0 7 8}
↦ {1 2 3 5}
↦ {1 2 4 6}
↦ {1 3 4 7}
↦ {2 5 6 8}
↦ {3 5 7 8}
↦ {4 6 7 8}
↦ {0 1 2 3 5} # Dimension 5
↦ {0 1 2 4 6}
↦ {0 1 3 4 7}
↦ {0 2 5 6 8}
↦ {0 3 5 7 8}
↦ {0 4 6 7 8}
↦
↦ MAXIMAL_CONES
↦ {0 1 2 3 5} # Dimension 5
↦ {0 1 2 4 6}
↦ {0 1 3 4 7}
↦ {0 2 5 6 8}
↦ {0 3 5 7 8}
↦ {0 4 6 7 8}
↦
groebnerComplex(I,number(7));
↦ _application PolyhedralFan
↦ _version 2.2
↦ _type PolyhedralFan
↦
↦ AMBIENT_DIM
↦ 5
↦
↦ DIM
↦ 5
↦
↦ LINEALITY_DIM
↦ 1
↦
↦ RAYS
↦ -1 0 0 0 0 # 0
↦ 0 -3 1 1 1 # 1
↦ 0 -1 -1 -1 3 # 2
↦ 0 -1 -1 3 -1 # 3
↦ 0 -1 3 -1 -1 # 4
↦ 0 1 -3 1 1 # 5
↦ 0 1 1 -3 1 # 6
↦ 0 1 1 1 -3 # 7
↦ 0 3 -1 -1 -1 # 8
↦
↦ N_RAYS
↦ 9
↦
↦ LINEALITY_SPACE
↦ 0 -1 -1 -1 -1 # 0
↦
↦ ORTH_LINEALITY_SPACE
↦ -1 0 0 0 0 # 0
↦ 0 1 -1 0 0 # 1
↦ 0 1 0 -1 0 # 2
↦ 0 1 0 0 -1 # 3
↦
↦ F_VECTOR
↦ 1 9 20 18 6
↦
↦ SIMPLICIAL
↦ 0
↦
↦ PURE
↦ 1
↦
↦ CONES
↦ {} # Dimension 1
↦ {0} # Dimension 2
↦ {1}
↦ {2}
↦ {3}
↦ {4}
↦ {5}
↦ {6}
↦ {7}
↦ {8}
↦ {0 1} # Dimension 3
↦ {0 2}
↦ {0 3}
↦ {0 4}
↦ {0 5}
↦ {0 6}
↦ {0 7}
↦ {0 8}
↦ {1 2}
↦ {1 3}
↦ {1 4}
↦ {2 5}
↦ {3 5}
↦ {2 6}
↦ {3 7}
↦ {4 6}
↦ {4 7}
↦ {5 8}
↦ {6 8}
↦ {7 8}
↦ {0 1 2} # Dimension 4
↦ {0 1 3}
↦ {0 1 4}
↦ {0 2 5}
↦ {0 3 5}
↦ {0 2 6}
↦ {0 3 7}
↦ {0 4 6}
↦ {0 4 7}
↦ {0 5 8}
↦ {0 6 8}
↦ {0 7 8}
↦ {1 2 3 5}
↦ {1 2 4 6}
↦ {1 3 4 7}
↦ {2 5 6 8}
↦ {3 5 7 8}
↦ {4 6 7 8}
↦ {0 1 2 3 5} # Dimension 5
↦ {0 1 2 4 6}
↦ {0 1 3 4 7}
↦ {0 2 5 6 8}
↦ {0 3 5 7 8}
↦ {0 4 6 7 8}
↦
↦ MAXIMAL_CONES
↦ {0 1 2 3 5} # Dimension 5
↦ {0 1 2 4 6}
↦ {0 1 3 4 7}
↦ {0 2 5 6 8}
↦ {0 3 5 7 8}
↦ {0 4 6 7 8}
↦
```

D.13.7 tropicalNewton_lib

Library: tropicalNewton.lib

Purpose: Computations in Tropical Geometry using Newton Polygon methods

Authors: Tommy Hofman, email: thofmann@mathematik.uni.kl.de Yue Ren, email: reny@cs.bgu.ac.il

Overview: This libraries contains algorithms for computing
- non-trivial points on tropical varieties,
- zero-dimensional tropical varieties,
- one-codimensional links of tropical varieties
based on Newton polygon methods.

References:

Hofmann, Ren: Computing tropical points and tropical links, arXiv:1611.02878 (WARNING: this library follows the max convention instead and triangular sets follow the definition of the Singular book)

Procedures: See also: Section D.13.6.39 [tropicalVariety], page 2178; Section D.13.6 [tropical_lib], page 2139.

D.13.7.1 setUniformizingParameter

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: setUniformizingParameter(p); p number

Return: none, sets the uniformizing parameter as p

Assume: char(K)==0 and p prime number or
trdeg(K)>0 and p transcendental variable or
p==0

Example:

```
LIB "tropicalNewton.lib";
// poor man's polynomials over Puiseux series:
ring r = (0,t),x,dp;
setUniformizingParameter(t);
val(t2+t3);
↦ 2
val(t^-2+t^-3);
↦ -3
// poor man's polynomials over p-adic numbers:
ring s = 0,x,dp;
setUniformizingParameter(2);
val(12);
↦ 2
val(1/12);
↦ -2
```

D.13.7.2 val

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: val(c); c number

Return: int, the valuation of a element in the ground field

Assume: uniformizingParameter is set and c!=0

Example:

```
LIB "tropicalNewton.lib";
// poor man's polynomials over Puiseux series:
ring r = (0,t),x,dp;
setUniformizingParameter(t);
val(t2+t3);
↦ 2
val(t^-2+t^-3);
↦ -3
// poor man's polynomials over p-adic numbers:
```

```
ring s = 0,x,dp;
setUniformizingParameter(2);
val(12);
↦ 2
val(1/12);
↦ -2
```

D.13.7.3 newtonPolygonNegSlopes

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: newtonPolygonNegSlopes(g,b); g poly, b int

Return: list, the negative slopes of the Newton Polygon of g
if b==1, computes root (type poly) instead if (easily) possible

Assume: uniformizingParameter is set and g univariate

Example:

```
LIB "tropicalNewton.lib";
ring r = (0,t),x,dp;
setUniformizingParameter(t);
poly g = tx2+x+1;
newtonPolygonNegSlopes(g);
↦ [1]:
↦ 0
↦ [2]:
↦ -1
// poor man's polynomials over p-adic numbers:
ring s = 0,x,dp;
setUniformizingParameter(3);
poly g = x2+9x+1;
newtonPolygonNegSlopes(g);
↦ [1]:
↦ 0
```

D.13.7.4 cccMatrixToPositiveIntvec

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: cccMatrixToPositiveIntvec(M); M matrix

Return: intvec, strictly positive equivalent as weight vector to row vector in M

Assume: constant coefficient case only,
will scale weight vector and add vectors of ones to it

Example:

```
LIB "tropicalNewton.lib";
ring r = (0,t),(p01,p02,p12,p03,p13,p23,p04,p14,p24,p34),dp;
number uniformizingParameter = t;
export(uniformizingParameter);
ideal I =
p23*p14-p13*p24+p12*p34,
p23*p04-p03*p24+p02*p34,
p13*p04-p03*p14+p01*p34,
p12*p04-p02*p14+p01*p24,
p12*p03-p02*p13+p01*p23;
system("--random",1337);
matrix p = tropicalPointNewton(I);
print(p);
↦ -59,-14,-87,57,-16,-63,41,-32,-90,-8
intvec w = cccMatrixToPositiveIntvec(p);
print(w);
↦ 32,
↦ 77,
↦ 4,
↦ 148,
↦ 75,
↦ 28,
↦ 132,
↦ 59,
↦ 1,
↦ 83
def s = switchRingsAndComputeInitialIdeal(I,w);
kill uniformizingParameter;
```

D.13.7.5 tropicalPointNewton

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: tropicalPointsLasVegas(I); I ideal

Return: matrix, a matrix containing a tropical point as row vector

Assume: uniformizingParameter is set and I monomial free

Note: if printlevel sufficiently high will print intermediate data and timings returns error if randomly chosen hyperplanes are not generic

Example:

```
LIB "tropicalNewton.lib";
ring r = (0,t),(p01,p02,p12,p03,p13,p23,p04,p14,p24,p34),dp;
number uniformizingParameter = t;
export(uniformizingParameter);
ideal I =
p23*p14-p13*p24+p12*p34,
p23*p04-p03*p24+p02*p34,
p13*p04-p03*p14+p01*p34,
p12*p04-p02*p14+p01*p24,
p12*p03-p02*p13+p01*p23;
system("--random",1337);
printlevel = 3;
matrix p = tropicalPointNewton(I);
↦ maximal independent set: 1,1,1,0,1,1,0,0,1,1
↦ substituting p01 with number of valuation 59
↦ substituting p02 with number of valuation 14
↦ substituting p12 with number of valuation 87
↦ substituting p13 with number of valuation 16
↦ substituting p23 with number of valuation 63
↦ substituting p24 with number of valuation 90
↦ substituting p34 with number of valuation 8
↦ computing triangular decomposition (picking first factor)
↦ starting analysis of Newton polygons
↦ possible valuations for p14 (picking first): (968*t^104+1870*t^95+4136*t^\
   89+7990*t^80+1738*t^74+3960*t^73+7650*t^64+7426*t^59-5658*t^58-3936*t^56-\
   3198*t^55+7110*t^43-2760*t^42-1920*t^40-1560*t^39-2760*t^35-1920*t^33-156\
   0*t^32)/(29*t^22+71)
↦ possible valuations for p04 (picking first): (-9889*t^175-14674*t^174-422\
   53*t^160-62698*t^159-13079*t^156-24211*t^153-35926*t^152-40455*t^144-6003\
   0*t^143-55883*t^141-103447*t^138-153502*t^137-32021*t^134-53505*t^125-990\
   45*t^122-146970*t^121-136817*t^119-130995*t^103+36300*t^102+70125*t^93+15\
   5100*t^87+299625*t^78+7744*t^75+68563*t^72+148500*t^71+14960*t^66+6545*t^\
   63+286875*t^62+33088*t^60+292951*t^57-212175*t^56-147600*t^54-119925*t^53\
   +63920*t^51+27965*t^48+13904*t^45+31680*t^44+6083*t^42+280485*t^41-103500\
   *t^40-72000*t^38-58500*t^37+61200*t^35-103500*t^33+26775*t^32-72000*t^31+\
   908*t^30-45264*t^29-5497*t^27-45387*t^26-13776*t^24-11193*t^23+56880*t^14\
   -22080*t^13+9525*t^11-22140*t^10-6720*t^8-5460*t^7-22080*t^6-15360*t^4-22\
   140*t^3-6720*t-5460)/(1189*t^86+580*t^70+2911*t^64+580*t^63+1420*t^48+142\
   0*t^41)
↦ possible valuations for p03 (picking first): (-899*t^133-1334*t^132-1189*\
   t^114-2201*t^111-3266*t^110-2911*t^92+3300*t^60+6375*t^51+704*t^33+6233*t\
   ^30+1360*t^24+595*t^21+1264*t^3+553)/(82*t^80+40*t^64+40*t^57)
↦ time used total: 0
↦ computing independent set: 0
↦ computing triangular decomposition: 0
↦ analyzing newton polygons: 0
print(p);
↦ -59,-14,-87,57,-16,-63,41,-32,-90,-8
intvec w = cccMatrixToPositiveIntvec(p);
print(w);
↦ 32,
↦ 77,
↦ 4,
↦ 148,
↦ 75,
↦ 28,
↦ 132,
↦ 59,
↦ 1,
↦ 83
def s = switchRingsAndComputeInitialIdeal(I,w);
↦ time used computing initial ideal: 0
kill uniformizingParameter;
```

D.13.7.6 switchRingsAndComputeInitialIdeal

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: switchRingsAndComputeInitialIdeal(I,w); I ideal, w intvec

Return: ring, a ring containing the initial ideal with respect to w

Assume: constant coefficient case and w strictly positive integer

Note: if printlevel sufficiently high will print timing

Example:

```
LIB "tropicalNewton.lib";
ring r = (0,t),(p01,p02,p12,p03,p13,p23,p04,p14,p24,p34),dp;
number uniformizingParameter = t;
export(uniformizingParameter);
ideal I =
p23*p14-p13*p24+p12*p34,
p23*p04-p03*p24+p02*p34,
p13*p04-p03*p14+p01*p34,
p12*p04-p02*p14+p01*p24,
p12*p03-p02*p13+p01*p23;
system("--random",1337);
printlevel = 3;
matrix p = tropicalPointNewton(I);
↦ maximal independent set: 1,1,1,0,1,1,0,0,1,1
↦ substituting p01 with number of valuation 59
↦ substituting p02 with number of valuation 14
↦ substituting p12 with number of valuation 87
↦ substituting p13 with number of valuation 16
↦ substituting p23 with number of valuation 63
↦ substituting p24 with number of valuation 90
↦ substituting p34 with number of valuation 8
↦ computing triangular decomposition (picking first factor)
↦ starting analysis of Newton polygons
↦ possible valuations for p14 (picking first): (968*t^104+1870*t^95+4136*t^\
   89+7990*t^80+1738*t^74+3960*t^73+7650*t^64+7426*t^59-5658*t^58-3936*t^56-\
   3198*t^55+7110*t^43-2760*t^42-1920*t^40-1560*t^39-2760*t^35-1920*t^33-156\
   0*t^32)/(29*t^22+71)
↦ possible valuations for p04 (picking first): (-9889*t^175-14674*t^174-422\
   53*t^160-62698*t^159-13079*t^156-24211*t^153-35926*t^152-40455*t^144-6003\
   0*t^143-55883*t^141-103447*t^138-153502*t^137-32021*t^134-53505*t^125-990\
   45*t^122-146970*t^121-136817*t^119-130995*t^103+36300*t^102+70125*t^93+15\
   5100*t^87+299625*t^78+7744*t^75+68563*t^72+148500*t^71+14960*t^66+6545*t^\
   63+286875*t^62+33088*t^60+292951*t^57-212175*t^56-147600*t^54-119925*t^53\
   +63920*t^51+27965*t^48+13904*t^45+31680*t^44+6083*t^42+280485*t^41-103500\
   *t^40-72000*t^38-58500*t^37+61200*t^35-103500*t^33+26775*t^32-72000*t^31+\
   908*t^30-45264*t^29-5497*t^27-45387*t^26-13776*t^24-11193*t^23+56880*t^14\
   -22080*t^13+9525*t^11-22140*t^10-6720*t^8-5460*t^7-22080*t^6-15360*t^4-22\
   140*t^3-6720*t-5460)/(1189*t^86+580*t^70+2911*t^64+580*t^63+1420*t^48+142\
   0*t^41)
↦ possible valuations for p03 (picking first): (-899*t^133-1334*t^132-1189*\
   t^114-2201*t^111-3266*t^110-2911*t^92+3300*t^60+6375*t^51+704*t^33+6233*t\
   ^30+1360*t^24+595*t^21+1264*t^3+553)/(82*t^80+40*t^64+40*t^57)
↦ time used total: 0
↦ computing independent set: 0
↦ computing triangular decomposition: 0
↦ analyzing newton polygons: 0
print(p);
↦ -59,-14,-87,57,-16,-63,41,-32,-90,-8
intvec w = cccMatrixToPositiveIntvec(p);
print(w);
↦ 32,
↦ 77,
↦ 4,
↦ 148,
↦ 75,
↦ 28,
↦ 132,
↦ 59,
↦ 1,
↦ 83
def s = switchRingsAndComputeInitialIdeal(I,w);
↦ time used computing initial ideal: 0
kill uniformizingParameter;
```

D.13.7.7 tropicalVarietyNewton

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: tropicalVarietyNewton(I); I ideal

Return: matrix, a matrix containing all elements of the tropical variety

Assume: uniformizingParameter is set, I monomial free and zero-dimensional

Example:

```
LIB "tropicalNewton.lib";
ring r = (0,t),(z,y,x),lp;
number uniformizingParameter = t;
export(uniformizingParameter);
ideal I = tx2+x+1,txy2+xy+1,xyz+1;
list TI = tropicalVarietyNewton(I);
for (int i=1; i<=size(TI); i++)
{ print(TI[i]); }
↦ 0,0,0
↦ 0,-1,1
↦ -1,1,0
↦ -2,1,1
kill uniformizingParameter;
```

D.13.7.8 tropicalLinkNewton

Procedure from library `tropicalNewton.lib` (see Section D.13.7 [tropicalNewton_lib], page 2195).

Usage: tropicalLinkNewton(inI); inI ideal

Return: matrix, a matrix containing generators of all rays of the tropical variety

Assume: constant coefficient case, inI is monomial free,
its tropical variety has codimension one lineality space and is a polyhedral fan

Note: if printlevel sufficiently high will print intermediate results

Example:

```
LIB "tropicalNewton.lib";
// a 10 valent facet in tropical Grass(3,7)
ring r = (0,t),
(p012,p013,p023,p123,p014,p024,p124,p034,p134,p234,
p015,p025,p125,p035,p135,p235,p045,p145,p245,p345,
p016,p026,p126,p036,p136,p236,p046,p146,p246,p346,
p056,p156,p256,p356,p456),
wp(4,7,5,7,4,4,4,7,5,7,2,1,2,4,4,4,2,1,2,4,7,5,7,7,
5,7,7,5,7,4,4,4,4,4,4);
number uniformizingParameter = t;
export(uniformizingParameter);
ideal inI =
p345*p136+p134*p356,  p125*p045+p015*p245,  p124*p015-p014*p125,
p135*p245-p125*p345,  p135*p045+p015*p345,  p124*p045+p014*p245,
p024*p125-p012*p245,  p145*p236-p124*p356,  p124*p135-p123*p145,
p024*p015+p012*p045,  p134*p026+p023*p146-p024*p136,
p145*p036+p014*p356,  p014*p135-p013*p145,  p234*p145+p124*p345,
p034*p145-p014*p345,  p024*p135-p012*p345,  p125*p035+p015*p235,
p235*p045-p035*p245,  p234*p136-p134*p236,  p134*p036-p034*p136,
p146*p356-p136*p456,  p135*p146-p134*p156,
p135*p026+p023*p156+p012*p356,  p124*p035+p014*p235,
p123*p025+p012*p235,  p013*p025-p012*p035,  p345*p146+p134*p456,
p125*p036+p015*p236,  p345*p026-p023*p456+p024*p356,
p123*p015-p013*p125,  p234*p025-p024*p235,  p034*p025-p024*p035,
p234*p125+p123*p245,  p245*p036-p045*p236,  p123*p045+p013*p245,
p034*p125-p013*p245,  p234*p015+p013*p245,  p245*p156+p125*p456,
p034*p015+p013*p045,  p045*p156-p015*p456,  p135*p236-p123*p356,
p235*p146-p134*p256,  p135*p036+p013*p356,  p124*p036+p014*p236,
p123*p014-p013*p124,  p035*p146-p134*p056,  p145*p126+p124*p156,
p234*p045-p034*p245,  p235*p026+p023*p256-p025*p236,
p145*p016+p014*p156,  p035*p026+p023*p056-p025*p036,
p345*p236+p234*p356,  p234*p135+p123*p345,  p345*p036+p034*p356,
p034*p135-p013*p345,  p345*p156+p135*p456,  p124*p034+p014*p234,
p145*p246-p124*p456,  p123*p024+p012*p234,  p145*p046+p014*p456,
p013*p024-p012*p034,  p024*p156+p012*p456,  p125*p056+p015*p256,
p245*p056-p045*p256,  p236*p146-p136*p246,  p134*p126+p123*p146,
p136*p046-p036*p146,  p235*p036-p035*p236,  p134*p016+p013*p146,
p123*p035+p013*p235,  p235*p156-p135*p256,
p123*p026-p023*p126+p012*p236,  p135*p056-p035*p156,
p023*p016-p013*p026+p012*p036,  p124*p056+p014*p256,
p234*p146-p134*p246,  p025*p126-p012*p256,  p134*p046-p034*p146,
p025*p016+p012*p056,  p234*p035-p034*p235,  p345*p256+p235*p456,
p234*p026+p023*p246-p024*p236,  p345*p056+p035*p456,
p034*p026+p023*p046-p024*p036,  p125*p016-p015*p126,
p025*p246-p024*p256,  p025*p046-p024*p056,  p245*p126-p125*p246,
p125*p046+p015*p246,  p045*p126+p015*p246,  p245*p016-p015*p246,
p045*p016-p015*p046,  p123*p036+p013*p236,  p236*p156+p126*p356,
p135*p126+p123*p156,  p036*p156-p016*p356,  p135*p016+p013*p156,
p124*p016-p014*p126,  p235*p056-p035*p256,  p245*p046-p045*p246,
p234*p036-p034*p236,  p123*p034+p013*p234,  p246*p356-p236*p456,
p234*p156-p123*p456,  p135*p246-p123*p456,  p345*p126-p123*p456,
p046*p356-p036*p456,  p034*p156+p013*p456,  p135*p046+p013*p456,
p345*p016-p013*p456,  p124*p046+p014*p246,  p024*p126-p012*p246,
p024*p016+p012*p046,  p345*p246+p234*p456,  p345*p046+p034*p456,
p235*p126+p123*p256,  p236*p056-p036*p256,  p123*p056+p013*p256,
p035*p126-p013*p256,  p235*p016+p013*p256,  p035*p016+p013*p056,
p235*p246-p234*p256,  p234*p056-p034*p256,  p035*p246-p034*p256,
p235*p046-p034*p256,  p035*p046-p034*p056,  p126*p036+p016*p236,
p123*p016-p013*p126,  p234*p126+p123*p246,  p236*p046-p036*p246,
p123*p046+p013*p246,  p034*p126-p013*p246,  p234*p016+p013*p246,
p246*p156+p126*p456,  p034*p016+p013*p046,  p046*p156-p016*p456,
p234*p046-p034*p246,  p126*p056+p016*p256,  p246*p056-p046*p256,
p126*p046+p016*p246,  p024*p235*p145+p124*p025*p345,
p024*p035*p145-p014*p025*p345,  p123*p145*p245-p124*p125*p345,
p013*p145*p245-p014*p125*p345,  p013*p045*p145+p014*p015*p345,
p024*p235*p136-p134*p025*p236,  p123*p245*p136+p134*p125*p236,
p013*p245*p136+p134*p015*p236,  p034*p245*p136-p134*p045*p236,
p134*p156*p356-p135*p136*p456,  p123*p145*p146-p124*p134*p156,
p013*p145*p146-p014*p134*p156,  p013*p145*p026+p023*p014*p156+p012*p014*p356,
p124*p025*p156+p012*p145*p256,  p012*p145*p056-p014*p025*p156,
p024*p145*p256-p124*p025*p456,  p024*p145*p056+p014*p025*p456,
p034*p235*p136-p134*p035*p236,  p134*p256*p356-p235*p136*p456,
p134*p056*p356-p035*p136*p456,  p025*p036*p146-p024*p136*p056,
p013*p125*p026-p023*p015*p126+p012*p015*p236,
p123*p245*p146+p134*p125*p246,  p013*p245*p146+p134*p015*p246,
p013*p245*p026-p023*p015*p246-p012*p045*p236,
p013*p045*p026-p023*p015*p046-p012*p045*p036,
p034*p245*p146-p134*p045*p246,  p013*p124*p026-p023*p014*p126+p012*p014*p236,
p013*p145*p056-p014*p035*p156,  p024*p256*p356-p025*p236*p456,
p024*p056*p356-p025*p036*p456,  p234*p256*p356-p235*p236*p456,
p034*p256*p356-p035*p236*p456,  p034*p056*p356-p035*p036*p456,
p012*p235*p145*p245+p124*p025*p125*p345,
p012*p035*p145*p245-p014*p025*p125*p345,
p012*p035*p045*p145+p014*p015*p025*p345,
p012*p235*p245*p136-p134*p025*p125*p236,
p012*p035*p245*p136+p134*p015*p025*p236,
p024*p035*p245*p136-p134*p025*p045*p236,
p014*p025*p125*p156+p012*p015*p145*p256,
p012*p145*p245*p256-p124*p025*p125*p456,
p012*p045*p145*p256+p014*p025*p125*p456,
p012*p245*p256*p356-p025*p125*p236*p456,
p012*p045*p256*p356+p015*p025*p236*p456,
p012*p045*p056*p356+p015*p025*p036*p456,
p123*p245*p256*p356+p125*p235*p236*p456,
p013*p245*p256*p356+p015*p235*p236*p456,
p013*p045*p256*p356+p015*p035*p236*p456,
p013*p045*p056*p356+p015*p035*p036*p456;
system("--random",1337);
printlevel = 3;
list TinI = tropicalLinkNewton(inI);
↦ reducing to one-dimensional fan
↦ intersecting with pairs of affine hyperplanes
↦ 1: empty
↦ 2: empty
↦ 3: empty
↦ 4: empty
↦ 5: empty
↦ 6: empty
↦ 7: empty
↦ 8: empty
↦ 9: empty
↦ 10: empty
↦ 11: non-empty, computing tropical variety
↦ total number of rays: 2
↦ 12: empty
↦ 13: non-empty, computing tropical variety
↦ total number of rays: 4
↦ 14: non-empty, computing tropical variety
↦ total number of rays: 6
↦ 15: non-empty, computing tropical variety
↦ total number of rays: 8
↦ 16: empty
↦ 17: empty
↦ 18: empty
↦ 19: empty
↦ 20: empty
↦ 21: empty
↦ 22: non-empty, computing tropical variety
↦ total number of rays: 10
↦ 23: empty
for (int i=1; i<=size(TinI); i++)
{ print(TinI[i]); }
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,1,-1,(-t+1),(-t+1),(t-1)\
  ,1,1,
↦   -1,0,1,1,-1,(t-1),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,-1,-1,_[1,24],_[1,25],_[\
  1,26],
↦   1,1,-1,0,1,1,-1,_[1,34],-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,(-t-1),-1,1,(-t),(t),1,1\
  ,-1,0,
↦   1,1,-1,(t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,_[1,22],-1,-1,-1/(t),1/(\
  t),1,1,
↦   -1,0,1,1,-1,1/(t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,(-t-1),-1,(-t),1,(t),1,1\
  ,-1,0,
↦   1,1,-1,(t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,_[1,22],-1,-1/(t),-1,1/(\
  t),1,1,
↦   -1,0,1,1,-1,1/(t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,(t-1),-1,(t),(t),1,1,1,-\
  1,0,1,
↦   1,-1,(-t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,_[1,22],-1,1/(t),1/(t),-\
  1,1,1,
↦   -1,0,1,1,-1,-1/(t),-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,(t-1),-1,(t),(t),(-t),1,\
  1,-1,0,
↦   1,1,-1,1,-1
↦ 0,0,0,0,0,0,-1,-1,0,1,0,0,-1,-1,0,1,1,-1,-1,-1,0,_[1,22],-1,1/(t),1/(t),-\
  1/(t),
↦   1,1,-1,0,1,1,-1,-1,-1
```

D.14 Miscellaneous libraries

D.14.1 arr_lib

Library: arr.lib

Purpose: a library of algorithms for arrangements of hyperplanes

Authors: Randolf Scholz (rscholz@rhrk.uni-kl.de),
Patrick Serwene (serwene@mathematik.uni-kl.de),
Lukas Kuehne (lf.kuehne@gmail.com)

Overloads:

// OPERATORS
= arrAdd assignment
+ arrAdd union of two arrs
[arrGet access to a single/multiple hyperplane(s) - arrMinus deletes given hyperplanes from the arr <= arrLEQ comparison
>= arrGEQ comparison
== arrEQ comparison
!= arrNEQ comparison
< arrLNEQ comparison
> arrGNEQ comparison

// TYPECASTING
matrix arr2mat coeff matrix
poly arr2poly defining polynomial

// OTHER
variables arrVariables ideal generated by the variables the arr depends on nvars arrNvars number of variables the arr depends on delete arrDelete deletes hyperplanes by indices print arrPrint prints the arr on the screen

// IDEAL INHERITED FUNCTIONS
homog arrHomog checks if arrangement is homogeneous simplify arrSimplify simplifies arrangement size arrSize number of planes
subst arrSubst substitute variables

// MULTI-ARRANGEMENTS
= multarrAdd assignment of multarr
+ multarrAdd union of multarr
poly multarr2poly defining polynomial
size multarrSize number of hyperplanes with mult. print multarrPrint displays multiarr
delete multarrDelete deletes hyperplane

Procedures:

D.14.1.1 arrSet

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrSet(A, k, p); arr A, int k, poly p;

Return: [arr] Arrangement where the k-th hyperplane is replaced by p.

Note: p must be linear

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
```

```
arr A = ideal(x,y,z);
arrSet(A,1,x+1);
↦ _[1]=x+1
↦ _[2]=y
↦ _[3]=z
↦
```

D.14.1.2 type2arr

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: type2arr(#); # def

Return: [arr] Arrangement defined by the input

Note: The procedure tries to cast the input to [arr] using arrAdd

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
ideal I = x,y,z;
typeof(type2arr(I));
↦ arr
```

D.14.1.3 mat2arr

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: mat2arr(M); matrix (M|b)

Return: [arr] interprets the rows of the matrix as the defining polynomial equations of the arrangement where the last column will be considered as the constant terms, i.e. if M is an m*(n+1) matrix we have
H_i = Ker(M_i1*x_1 +...+ M_in*x_n + M_i(n+1)) for i=1...m and A = {H_1,...,H_m} the resulting arrangement.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
matrix M[4][4] = 1,0,1,1,1,1,0,2,0,1,1,3,2,1,1,4;
print(M);
↦ 1,0,1,1,
↦ 1,1,0,2,
↦ 0,1,1,3,
↦ 2,1,1,4
mat2arr(M);
↦ _[1]=x+z+1
↦ _[2]=x+y+2
↦ _[3]=y+z+3
↦ _[4]=2x+y+z+4
↦
```

See also: Section D.14.1.4 [mat2carr], page 2207.

D.14.1.4 mat2carr

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: mat2carr(M); matrix M

Return: [arr] interprets the rows of the matrix as the defining polynomial equations of the arrangement. I.e. if M is an m*n matrix we have H_i = Ker(M_i1*x_1 +...+ M_in*x_n) for i=1...m and
A = {H_1,...,H_m} the resulting arrangement.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
matrix M[4][3] = 1,0,1,1,1,0,0,1,1,2,1,1;
print(M);
↦ 1,0,1,
↦ 1,1,0,
↦ 0,1,1,
↦ 2,1,1
mat2carr(M);
↦ _[1]=x+z
↦ _[2]=x+y
↦ _[3]=y+z
↦ _[4]=2x+y+z
↦
```

See also: Section D.14.1.3 [mat2arr], page 2206.

D.14.1.5 arrPrintMatrix

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arPrintMatrix(arr A)

Return: [] prints arr in matrix form

Note: differs print(matrix(arr A)) since variables included

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A = mat2arr(random(20,5,4));
A;
↦ _[1]=10x+y-2z-12
↦ _[2]=2x-16z-14
↦ _[3]=11y+16z-5
↦ _[4]=-16x-18y-5z+11
↦ _[5]=3x+8y+6
↦
arrPrintMatrix(A);
↦ 10x, y,   -2z, -12,
↦ 2x,  0,   -16z,-14,
↦ 0,   11y, 16z, -5,
↦ -16x,-18y,-5z, 11,
↦ 3x,  8y,  0,   6
```

D.14.1.6 varMat

Procedure from library arr.lib (see Section D.14.1 [arr_lib], page 2205).

Usage: varMat(v); v intvec

Return: [matrix] M containing the corresponding ring_variables

Example:

```
LIB "arr.lib";
ring R = 0,(x(1..6)),dp;
intvec v = 2,4,6;
varMat(v);
↦ _[1,1]=x(2)
↦ _[2,1]=x(4)
↦ _[3,1]=x(6)
```

See also: Section D.14.1.9 [arrLastVar], page 2209; Section D.14.1.8 [arrSwapVar], page 2208; Section D.14.1.6 [varMat], page 2208; Section D.14.1.7 [varNum], page 2208.

D.14.1.7 varNum

Procedure from library arr.lib (see Section D.14.1 [arr_lib], page 2205).

Usage: varnum(string s);
varnum(ring_variable);

Return: [int] number of given ring variable, or 0 if it does not appear

Note: This procedure has the same functionality as varNum from the dmod.lib package, but also accepts polys as input.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
varNum(y);
↦ 2
ring S = 0,(x(1..5),y(1..5)),dp;
varNum("y(3)");
↦ 8
```

See also: Section D.14.1.9 [arrLastVar], page 2209; Section D.14.1.8 [arrSwapVar], page 2208; Section D.14.1.6 [varMat], page 2208; Section D.14.1.7 [varNum], page 2208.

D.14.1.8 arrSwapVar

Procedure from library arr.lib (see Section D.14.1 [arr_lib], page 2205).

Usage: arrSwapVar(A, i, j); arr A, ring_variables/integers i,j

Return: [arr] A where variables i and j are swapped.

Note: if i and/or j are integers the algorithm considers the variables variables(A)[i] and/or variables(A)[j]

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),lp;
arr A  = ideal(x+1,x+y,z);
```

```
arrSwapVar(A,x,z);
↦ _[1]=z+1
↦ _[2]=y+z
↦ _[3]=x
↦
```

See also: Section D.14.1.9 [arrLastVar], page 2209; Section D.14.1.8 [arrSwapVar], page 2208; Section D.14.1.6 [varMat], page 2208; Section D.14.1.7 [varNum], page 2208.

D.14.1.9 arrLastVar

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrLastVar(A); arr A

Return: [int] number of the last variable A uses

Note: useful if you want a list containing all variables x_1 ... x_k used in A, but you do not want to skip any like variables(A) does.

Example:

```
LIB "arr.lib";
ring R = 0,x(1..10),dp;
arr A = ideal(x(1), x(2), x(3), x(6));
int n = arrLastVar(A);
varMat(1..n);
↦ _[1,1]=x(1)
↦ _[2,1]=x(2)
↦ _[3,1]=x(3)
↦ _[4,1]=x(4)
↦ _[5,1]=x(5)
↦ _[6,1]=x(6)
variables(A);
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ _[3]=x(3)
↦ _[4]=x(6)
```

See also: Section D.14.1.9 [arrLastVar], page 2209; Section D.14.1.8 [arrSwapVar], page 2208; Section D.14.1.6 [varMat], page 2208; Section D.14.1.7 [varNum], page 2208.

D.14.1.10 arrCenter

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCenter(A); arr A

Return: [list] L entry 0 if A not centered or entries 1, x, H, where x is any particular point of the center and H is a matrix consisting of vectors which spanning linear intersection space.
If there is exactly one solution, then H = 0.

Note: The intersection of all hyperplanes can be expressed in forms of a linear system Ax=b, where (A|b) is the coeff. matrix of the arrange- ment, which is then solved using L-U decomposition

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A= ideal(x,y,x-y+1);    // centerless
arrCenter(A);
↦ [1]:
↦    0
arr B= ideal(x,y,z);        // center is a single point
arrCenter(B);
↦ [1]:
↦    1
↦ [2]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=0
↦ [3]:
↦    _[1,1]=0
arr C= ideal(x,z,x+z);      // center is a line
// here we get a wrong result because the matrix is simplified since A doesn't
// contain any "y" the matrix (A|b) will be 3x3 only.
arrCenter(C);
↦ [1]:
↦    1
↦ [2]:
↦    _[1,1]=0
↦    _[2,1]=0
↦    _[3,1]=0
↦ [3]:
↦    _[1,1]=0
↦    _[2,1]=-1
↦    _[3,1]=0
```

See also: Section D.14.1.10 [arrCenter], page 2209; Section D.14.1.12 [arrCentered], page 2211; Section D.14.1.11 [arrCentral], page 2210; Section D.14.1.13 [arrCentralize], page 2211.

D.14.1.11 arrCentral

Procedure from library arr.lib (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCentral(A); arr A

Return: [0,1] true if arr is central(i.e. all planes intersect in 0)

Note: This is the same as homog(A)

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
// centered and central
arr A = ideal(x,y,z);
arrCentered(A);
↦ 1
arrCentral(A);
↦ 1
// centered but not central (center: (-1,-1/2, 1))
arr B = ideal(x+1,2y+1,-z+1);
arrCentered(B);
```

```
↦ 1
arrCentral(B);
↦ 0
```

See also: Section D.14.1.10 [arrCenter], page 2209; Section D.14.1.12 [arrCentered], page 2211; Section D.14.1.11 [arrCentral], page 2210; Section D.14.1.13 [arrCentralize], page 2211.

D.14.1.12 arrCentered

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCentered(A); arr A

Return: [0,1] true if A is centered(i.e. intersection of all planes not empty)

Note: The algorithm uses the rank of matrix: Ax=b has a solution iff rank(A) = rank(A|b)

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A= ideal(x,y,x-y+1);        // centerless
arrCentral(A);
↦ 0
arr B= ideal(x,y,z);            // central with center being the origin
arrCentral(B);
↦ 1
arr C= ideal(x+1,2y+1,-z+1);    // central with center (-1,-1/2, 1)
arrCentral(C);
↦ 0
```

See also: Section D.14.1.10 [arrCenter], page 2209; Section D.14.1.12 [arrCentered], page 2211; Section D.14.1.11 [arrCentral], page 2210; Section D.14.1.13 [arrCentralize], page 2211.

D.14.1.13 arrCentralize

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCenteralize(A); arr A

Return: [arr] A after centralization via coordinate change

Note: The coordinate change only does translation, vector of translation is the second output of arrCenter

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A = ideal(x-1,y,x-z-1,x-z-1);
arrCentralize(A);
↦ _[1]=x-2
↦ _[2]=y
↦ _[3]=x-z-2
↦
```

See also: Section D.14.1.10 [arrCenter], page 2209; Section D.14.1.12 [arrCentered], page 2211; Section D.14.1.11 [arrCentral], page 2210; Section D.14.1.13 [arrCentralize], page 2211.

D.14.1.14 arrCoordChange

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCoordChange(A, T); arr A, (m*n mat) n*n or n*n+1 matrix T arrCoordChange(A, T , c); arr A, n*n matrix T, n*1 matrix/vector

Return: [arr]: Arrangement A [A|b] after a coordinate change f: x -> Tx + c with T invertible i.e. [A|b] => [AT^-1|b+AT^-1c] since we have
f(H) = f(ker(a1*x1 + ... + an*xn - b)) = {f(x) : a'x -b = 0} = {y : a'f^-1(y) -b = 0)}
= {y : a'(T^-1(y-c)) - b = 0}
= {y : a'T^-1y -(b + a'T^-1c) = 0}

Note: There are 3 options how you can give the input (in each case n <= nvars(basering)) 1. Just a nxn matrix with
=> Will automatically complete T by a unit matrix and perform x -> Tx 2. A nxn matrix T and a nx1 vector/matrix c with
=> Will automatically complete T and c and perform x -> Tx +c 3. A nxn+1 matrix T with
=> will use last column as translation vector c

Example:

```
LIB "arr.lib";
ring r = 0,(x,y,z),lp;
arr A = x,y,z;
arrCoordChange(A,1,[0,0,1]); //lifts z-hyperplane by 1 unit
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z-1
↦ 
matrix T[2][2] = [0,1,1,0];  // swaps x and y
arrCoordChange(A,T);
↦ _[1]=y
↦ _[2]=x
↦ _[3]=z
↦ 
matrix c[2][1] = [1,0];
T = concat(T,c);            // now swap x and y and add 1 to x afterwards
arrCoordChange(A,T);
↦ _[1]=y
↦ _[2]=x-1
↦ _[3]=z
↦ 
// Note how T doesn't even need to be a full 3x3 base change matrix.
```

See also: Section D.14.1.14 [arrCoordChange], page 2212; Section D.14.1.15 [arrCoordNormalize], page 2212.

D.14.1.15 arrCoordNormalize

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCoordChange(A, v);

Return: [arr]: Arrangement after a coordinate change that transforms the arrangement such that after a transformation x -> Tx + c we have the arrangement has the matrix representation [AT^-1|b+AT^-1c] such that [AT^-1]_v = I and [b+AT^-1c]_v = 0;

Note: algorithm performs a base change if H_k is homogeneous (i.e. has no) constant term and an affine transformation otherwise
Ax+b = 0, Transformation x = Ty+c: AT^-1y + AT^-1c + b = 0 Now we want to have (AT^-1)_v = I and (AT^-1c +b)_v = AT^-1_v*c + b_v = 0

Example:

```
LIB "arr.lib";
ring r = 0,(x,y,z),lp;
arr A = ideal(x,y,z,x+z+4);
intvec v = 1,2,4;
arrCoordNormalize(A,v);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=-x+z-4
↦ _[4]=z
↦
```

See also: Section D.14.1.14 [arrCoordChange], page 2212; Section D.14.1.15 [arrCoordNormalize], page 2212.

D.14.1.16 arrCone

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCone(A);
arrCone(A, ring_variable); arr A arrangement in variables x_1...x_n;

Return: arr, the coned hyperplane Arrangement cA with respect to the given ring_variable, or the last ring_variable if none was given.

Note: The hyperplanes are homogenized w.r.t. v and a new hyperplane H = ker(x_n+1) is added.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr  A = ideal(x+1, x,x-2,x-1);
arrCone(A, y);
↦ _[1]=x+y
↦ _[2]=x
↦ _[3]=x-2y
↦ _[4]=x-y
↦ _[5]=y
↦
arr B= ideal(x,y,x+y-1);
arrCone(B);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=x+y-z
↦ _[4]=z
↦
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.17 arrDecone

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrDecone(A, k); arrangement A, integer k;

Return: arr: the deconed hyperplane Arrangement dA

Note: A has to be non-empty and central. arrDecone is an inverse operation to arrCone since A == arrDecone(arrCone(A),size(A)+1) for any A. One can also decone a central arrangement with respect to any hyper- plane k, but than a coordinate change is necessary to make H_k = ker(x_k). Since such a coordinate change is not unique, use arrCoordchange to do so.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A= ideal(x,y,z,x+y-z);
arrDecone(A,3);
↦ _[1]=x
↦ _[2]=y
↦ _[3]=x+y-1
↦
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.18 arrLocalize

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrLocalize(A, v); arrangement A, intvec v;

Return: arr: the localized arrangement A_X, i.e. A_X only contains the hyperplanes which contain the flat X, which is defined by the equations A[v]

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A=arrTypeB(3);
intvec v=5,8;
arr B=arrLocalize(A,v);
B;
↦ _[1]=x-y
↦ _[2]=x+y
↦ _[3]=x
↦ _[4]=y
↦
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.19 arrRestrict

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Return: arr: the restricted hyperplane Arrangement (A^X)

Note: A has to be non-empty.

Remarks: We restrict A to the flat X, defined by the equations in A[v]. The restriction will only be performed, if the ideal defining the flat X is monomial (i.e. X is an intersection of coordinate planes). If the optional argument CC is given, the arrangement is transformed in such a way that X has the above form.

Example:

```
LIB "arr.lib";
ring S = 0,(x,y,z),dp;
arr A  = arrTypeB(3);
A;
↦ _[1]=x-y
↦ _[2]=x+y
↦ _[3]=x-z
↦ _[4]=x+z
↦ _[5]=x
↦ _[6]=y-z
↦ _[7]=y+z
↦ _[8]=y
↦ _[9]=z
↦
arrRestrict(A,9);
↦ _[1]=x-y
↦ _[2]=x+y
↦ _[3]=x
↦ _[4]=y
↦
arrRestrict(A,4,"CC");
↦ _[1]=1/2y-z
↦ _[2]=1/2y+z
↦ _[3]=y
↦ _[4]=z
↦
intvec v=5,8;
arrRestrict(A,v);
↦ _[1]=-z
↦
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.20 arrIsEssential

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrIsEssential(A); arrangement A;

Return: boolean: 1 if arr is essential, i.e. rank of maximal element of poset is dimension

Note: A has to be non-empty.

Example:

```
LIB "arr.lib";
ring S = 0,(x,y,z),lp;
arr A = ideal(x,y,z);
arr B = ideal(x+y+z,x,y+z);
arrIsEssential(A);
↦ 1
arrIsEssential(B);
↦ 0
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.21 arrEssentialize

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrEssentialize(A); arrangement A;

Return: essential arrangement by transformation

Note: A has to be non-empty.

Example:

```
LIB "arr.lib";
ring S = 0,(x,y,z),dp;
arr A=arrBraid(3);
arrEssentialize(A);
↦ _[1]=x-y
↦ _[2]=x
↦ _[3]=y
↦
arr B = ideal(x+y+z,x,y+z);
arrEssentialize(B);
↦ _[1]=x+y
↦ _[2]=x
↦ _[3]=y
↦
```

See also: Section D.14.1.16 [arrCone], page 2213; Section D.14.1.17 [arrDecone], page 2214; Section D.14.1.21 [arrEssentialize], page 2216; Section D.14.1.20 [arrIsEssential], page 2215; Section D.14.1.19 [arrRestrict], page 2215.

D.14.1.22 arrBoolean

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrBoolean(v); int v

Return: arr, which uses the first v variables of ring for boolean arrangement

Example:

```
LIB "arr.lib";
ring R = 0,x(1..10),dp;
```

```
arrBoolean(7);
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ _[3]=x(3)
↦ _[4]=x(4)
↦ _[5]=x(5)
↦ _[6]=x(6)
↦ _[7]=x(7)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.23 arrBraid

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrBraid(v); int v

Return: Type A (braid) arrangement of dimension v

Example:

```
LIB "arr.lib";
ring R = 0,x(1..10),dp;
arrBraid(7);
↦ _[1]=x(1)-x(2)
↦ _[2]=x(1)-x(3)
↦ _[3]=x(1)-x(4)
↦ _[4]=x(1)-x(5)
↦ _[5]=x(1)-x(6)
↦ _[6]=x(1)-x(7)
↦ _[7]=x(2)-x(3)
↦ _[8]=x(2)-x(4)
↦ _[9]=x(2)-x(5)
↦ _[10]=x(2)-x(6)
↦ _[11]=x(2)-x(7)
↦ _[12]=x(3)-x(4)
↦ _[13]=x(3)-x(5)
↦ _[14]=x(3)-x(6)
↦ _[15]=x(3)-x(7)
↦ _[16]=x(4)-x(5)
↦ _[17]=x(4)-x(6)
↦ _[18]=x(4)-x(7)
↦ _[19]=x(5)-x(6)
↦ _[20]=x(5)-x(7)
↦ _[21]=x(6)-x(7)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.24 arrTypeB

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrTypeB(v); int v

Return: arrangement, which uses first v variables of ring for reflection arrangement of type B

Example:

```
LIB "arr.lib";
ring R = 0,x(1..10),dp;
arrTypeB(5);
↦ _[1]=x(1)-x(2)
↦ _[2]=x(1)+x(2)
↦ _[3]=x(1)-x(3)
↦ _[4]=x(1)+x(3)
↦ _[5]=x(1)-x(4)
↦ _[6]=x(1)+x(4)
↦ _[7]=x(1)-x(5)
↦ _[8]=x(1)+x(5)
↦ _[9]=x(1)
↦ _[10]=x(2)-x(3)
↦ _[11]=x(2)+x(3)
↦ _[12]=x(2)-x(4)
↦ _[13]=x(2)+x(4)
↦ _[14]=x(2)-x(5)
↦ _[15]=x(2)+x(5)
↦ _[16]=x(2)
↦ _[17]=x(3)-x(4)
↦ _[18]=x(3)+x(4)
↦ _[19]=x(3)-x(5)
↦ _[20]=x(3)+x(5)
↦ _[21]=x(3)
↦ _[22]=x(4)-x(5)
↦ _[23]=x(4)+x(5)
↦ _[24]=x(4)
↦ _[25]=x(5)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.25 arrTypeD

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrTypeD(v); int v

Return: arrangement, which uses first v variables of ring for reflection arrangement of type D

Example:

```
LIB "arr.lib";
ring R = 0,x(1..10),dp;
arrTypeD(5);
↦ _[1]=x(1)-x(2)
↦ _[2]=x(1)+x(2)
↦ _[3]=x(1)-x(3)
↦ _[4]=x(1)+x(3)
↦ _[5]=x(1)-x(4)
↦ _[6]=x(1)+x(4)
↦ _[7]=x(1)-x(5)
↦ _[8]=x(1)+x(5)
↦ _[9]=x(2)-x(3)
↦ _[10]=x(2)+x(3)
↦ _[11]=x(2)-x(4)
↦ _[12]=x(2)+x(4)
↦ _[13]=x(2)-x(5)
↦ _[14]=x(2)+x(5)
↦ _[15]=x(3)-x(4)
↦ _[16]=x(3)+x(4)
↦ _[17]=x(3)-x(5)
↦ _[18]=x(3)+x(5)
↦ _[19]=x(4)-x(5)
↦ _[20]=x(4)+x(5)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.26 arrRandom

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrRandom(n,v,N); int n,v,N

Return: Random arrangement, where m is the number of hyperplanes, n the dimension, d the upper bound for absolute value of coefficients.

Note: You can also write arr = random(d,m,n) to create random arrangements

Example:

```
LIB "arr.lib";
ring R = 0,x(1..20),dp;
arrRandom(7,3,15);
↦ _[1]=2*x(1)-6*x(2)-3*x(3)+4*x(5)+x(6)-4*x(7)-3*x(8)-x(9)+x(10)-3*x(11)-6*\
   x(12)-x(13)+2*x(14)-6*x(15)-4
↦ _[2]=x(1)-3*x(2)-4*x(3)-7*x(5)-3*x(6)+x(8)-3*x(9)-4*x(10)-x(11)-7*x(12)-2\
   *x(13)-3*x(14)-1
↦ _[3]=4*x(1)+2*x(2)+5*x(3)-x(4)+6*x(5)+5*x(6)-6*x(7)+5*x(8)+x(10)+3*x(11)-\
   5*x(12)+2*x(13)+6*x(14)+x(15)+3
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.27 arrRandomCentral

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrRandomCentral(d,m,n); int d,m,n

Return: Random central arrangement, where m is the number of hyperplanes, n the dimension, d the upper bound for absolute value of coefficients.

Example:

```
LIB "arr.lib";
ring R = 0,x(1..20),dp;
arrRandomCentral(7,3,15);
↦ _[1]=2*x(1)-6*x(2)-3*x(3)+4*x(5)+x(6)-4*x(7)-3*x(8)-x(9)+x(10)-3*x(11)-6*\
   x(12)-x(13)+2*x(14)-6*x(15)
↦ _[2]=-4*x(1)+x(2)-3*x(3)-4*x(4)-7*x(6)-3*x(7)+x(9)-3*x(10)-4*x(11)-x(12)-\
   7*x(13)-2*x(14)-3*x(15)
↦ _[3]=-x(2)+4*x(3)+2*x(4)+5*x(5)-x(6)+6*x(7)+5*x(8)-6*x(9)+5*x(10)+x(12)+3\
   *x(13)-5*x(14)+2*x(15)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.28 arrEdelmanReiner

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrEdelmanReiner();

Return: the Edelman-Reiner arrangement, which is a free arrangement but the restriction to the 6-th hyperplane is nonfree.
(i.e. counterexample for Orlik-Conjecture)

Note: the active ring must have at least five variables

Example:

```
LIB "arr.lib";
ring r=0,x(1..5),dp;
arrEdelmanReiner();
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ _[3]=x(3)
↦ _[4]=x(4)
↦ _[5]=x(5)
↦ _[6]=x(1)-x(2)-x(3)-x(4)-x(5)
↦ _[7]=x(1)-x(2)-x(3)-x(4)+x(5)
↦ _[8]=x(1)-x(2)-x(3)+x(4)-x(5)
↦ _[9]=x(1)-x(2)-x(3)+x(4)+x(5)
↦ _[10]=x(1)-x(2)+x(3)-x(4)-x(5)
↦ _[11]=x(1)-x(2)+x(3)-x(4)+x(5)
↦ _[12]=x(1)-x(2)+x(3)+x(4)-x(5)
↦ _[13]=x(1)-x(2)+x(3)+x(4)+x(5)
↦ _[14]=x(1)+x(2)-x(3)-x(4)-x(5)
↦ _[15]=x(1)+x(2)-x(3)-x(4)+x(5)
↦ _[16]=x(1)+x(2)-x(3)+x(4)-x(5)
↦ _[17]=x(1)+x(2)-x(3)+x(4)+x(5)
↦ _[18]=x(1)+x(2)+x(3)-x(4)-x(5)
↦ _[19]=x(1)+x(2)+x(3)-x(4)+x(5)
↦ _[20]=x(1)+x(2)+x(3)+x(4)-x(5)
↦ _[21]=x(1)+x(2)+x(3)+x(4)+x(5)
↦
```

See also: Section D.14.1.22 [arrBoolean], page 2216; Section D.14.1.23 [arrBraid], page 2217; Section D.14.1.28 [arrEdelmanReiner], page 2220; Section D.14.1.26 [arrRandom], page 2219; Section D.14.1.24 [arrTypeB], page 2217; Section D.14.1.25 [arrTypeD], page 2218.

D.14.1.29 arrOrlikSolomon

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrOrlikSolomon(A); arr A

Return: [ring] exterior Algebra E as ring with Orlik-Solomon ideal as attribute I. The Orlik-Solomon ideal is generated by the differentials of dependent tuples of hyperplanes. For a complex arrangement the quotient E/I is isomorphic to the cohomology ring of the complement of the arrangement.

Note: In order to access this ideal I activate this exterior algebra with setring.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A = arrTypeB(3);
def E = arrOrlikSolomon(A);
setring E;
//The generators of the Orlik-Solomon-Ideal are:
I;
↦ I[1]=e(7)*e(8)-e(7)*e(9)+e(8)*e(9)
↦ I[2]=e(6)*e(8)-e(6)*e(9)+e(8)*e(9)
↦ I[3]=e(6)*e(7)-e(6)*e(9)+e(7)*e(9)
↦ I[4]=e(4)*e(5)-e(4)*e(9)+e(5)*e(9)
↦ I[5]=e(3)*e(5)-e(3)*e(9)+e(5)*e(9)
↦ I[6]=e(3)*e(4)-e(3)*e(9)+e(4)*e(9)
↦ I[7]=e(6)*e(7)-e(6)*e(8)+e(7)*e(8)
↦ I[8]=e(2)*e(5)-e(2)*e(8)+e(5)*e(8)
↦ I[9]=e(1)*e(5)-e(1)*e(8)+e(5)*e(8)
↦ I[10]=e(1)*e(2)-e(1)*e(8)+e(2)*e(8)
↦ I[11]=e(1)*e(4)-e(1)*e(7)+e(4)*e(7)
↦ I[12]=e(2)*e(3)-e(2)*e(7)+e(3)*e(7)
↦ I[13]=e(2)*e(4)-e(2)*e(6)+e(4)*e(6)
↦ I[14]=e(1)*e(3)-e(1)*e(6)+e(3)*e(6)
↦ I[15]=e(3)*e(4)-e(3)*e(5)+e(4)*e(5)
↦ I[16]=e(1)*e(2)-e(1)*e(5)+e(2)*e(5)
↦ I[17]=-e(2)*e(4)*e(8)+e(2)*e(4)*e(9)-e(2)*e(8)*e(9)+e(4)*e(8)*e(9)
↦ I[18]=-e(1)*e(4)*e(8)+e(1)*e(4)*e(9)-e(1)*e(8)*e(9)+e(4)*e(8)*e(9)
↦ I[19]=-e(2)*e(3)*e(8)+e(2)*e(3)*e(9)-e(2)*e(8)*e(9)+e(3)*e(8)*e(9)
↦ I[20]=-e(1)*e(3)*e(8)+e(1)*e(3)*e(9)-e(1)*e(8)*e(9)+e(3)*e(8)*e(9)
↦ I[21]=-e(2)*e(5)*e(7)+e(2)*e(5)*e(9)-e(2)*e(7)*e(9)+e(5)*e(7)*e(9)
↦ I[22]=-e(1)*e(5)*e(7)+e(1)*e(5)*e(9)-e(1)*e(7)*e(9)+e(5)*e(7)*e(9)
↦ I[23]=-e(2)*e(4)*e(7)+e(2)*e(4)*e(9)-e(2)*e(7)*e(9)+e(4)*e(7)*e(9)
↦ I[24]=-e(1)*e(3)*e(7)+e(1)*e(3)*e(9)-e(1)*e(7)*e(9)+e(3)*e(7)*e(9)
↦ I[25]=-e(1)*e(2)*e(7)+e(1)*e(2)*e(9)-e(1)*e(7)*e(9)+e(2)*e(7)*e(9)
↦ I[26]=-e(2)*e(5)*e(6)+e(2)*e(5)*e(9)-e(2)*e(6)*e(9)+e(5)*e(6)*e(9)
↦ I[27]=-e(1)*e(5)*e(6)+e(1)*e(5)*e(9)-e(1)*e(6)*e(9)+e(5)*e(6)*e(9)
↦ I[28]=-e(1)*e(4)*e(6)+e(1)*e(4)*e(9)-e(1)*e(6)*e(9)+e(4)*e(6)*e(9)
↦ I[29]=-e(2)*e(3)*e(6)+e(2)*e(3)*e(9)-e(2)*e(6)*e(9)+e(3)*e(6)*e(9)
↦ I[30]=-e(1)*e(2)*e(6)+e(1)*e(2)*e(9)-e(1)*e(6)*e(9)+e(2)*e(6)*e(9)
↦ I[31]=-e(1)*e(2)*e(4)+e(1)*e(2)*e(9)-e(1)*e(4)*e(9)+e(2)*e(4)*e(9)
↦ I[32]=-e(1)*e(2)*e(3)+e(1)*e(2)*e(9)-e(1)*e(3)*e(9)+e(2)*e(3)*e(9)
↦ I[33]=-e(4)*e(5)*e(7)+e(4)*e(5)*e(8)-e(4)*e(7)*e(8)+e(5)*e(7)*e(8)
↦ I[34]=-e(3)*e(5)*e(7)+e(3)*e(5)*e(8)-e(3)*e(7)*e(8)+e(5)*e(7)*e(8)
↦ I[35]=-e(3)*e(4)*e(7)+e(3)*e(4)*e(8)-e(3)*e(7)*e(8)+e(4)*e(7)*e(8)
↦ I[36]=-e(2)*e(4)*e(7)+e(2)*e(4)*e(8)-e(2)*e(7)*e(8)+e(4)*e(7)*e(8)
↦ I[37]=-e(1)*e(3)*e(7)+e(1)*e(3)*e(8)-e(1)*e(7)*e(8)+e(3)*e(7)*e(8)
↦ I[38]=-e(4)*e(5)*e(6)+e(4)*e(5)*e(8)-e(4)*e(6)*e(8)+e(5)*e(6)*e(8)
↦ I[39]=-e(3)*e(5)*e(6)+e(3)*e(5)*e(8)-e(3)*e(6)*e(8)+e(5)*e(6)*e(8)
↦ I[40]=-e(3)*e(4)*e(6)+e(3)*e(4)*e(8)-e(3)*e(6)*e(8)+e(4)*e(6)*e(8)
↦ I[41]=-e(1)*e(4)*e(6)+e(1)*e(4)*e(8)-e(1)*e(6)*e(8)+e(4)*e(6)*e(8)
↦ I[42]=-e(2)*e(3)*e(6)+e(2)*e(3)*e(8)-e(2)*e(6)*e(8)+e(3)*e(6)*e(8)
↦ I[43]=-e(2)*e(3)*e(4)+e(2)*e(3)*e(8)-e(2)*e(4)*e(8)+e(3)*e(4)*e(8)
↦ I[44]=-e(1)*e(3)*e(4)+e(1)*e(3)*e(8)-e(1)*e(4)*e(8)+e(3)*e(4)*e(8)
↦ I[45]=-e(4)*e(5)*e(6)+e(4)*e(5)*e(7)-e(4)*e(6)*e(7)+e(5)*e(6)*e(7)
↦ I[46]=-e(3)*e(5)*e(6)+e(3)*e(5)*e(7)-e(3)*e(6)*e(7)+e(5)*e(6)*e(7)
↦ I[47]=-e(2)*e(5)*e(6)+e(2)*e(5)*e(7)-e(2)*e(6)*e(7)+e(5)*e(6)*e(7)
↦ I[48]=-e(1)*e(5)*e(6)+e(1)*e(5)*e(7)-e(1)*e(6)*e(7)+e(5)*e(6)*e(7)
↦ I[49]=-e(3)*e(4)*e(6)+e(3)*e(4)*e(7)-e(3)*e(6)*e(7)+e(4)*e(6)*e(7)
↦ I[50]=-e(1)*e(2)*e(6)+e(1)*e(2)*e(7)-e(1)*e(6)*e(7)+e(2)*e(6)*e(7)
↦ I[51]=-e(2)*e(4)*e(5)+e(2)*e(4)*e(7)-e(2)*e(5)*e(7)+e(4)*e(5)*e(7)
↦ I[52]=-e(1)*e(3)*e(5)+e(1)*e(3)*e(7)-e(1)*e(5)*e(7)+e(3)*e(5)*e(7)
↦ I[53]=-e(1)*e(4)*e(5)+e(1)*e(4)*e(6)-e(1)*e(5)*e(6)+e(4)*e(5)*e(6)
↦ I[54]=-e(2)*e(3)*e(5)+e(2)*e(3)*e(6)-e(2)*e(5)*e(6)+e(3)*e(5)*e(6)
↦ I[55]=-e(1)*e(2)*e(3)+e(1)*e(2)*e(4)-e(1)*e(3)*e(4)+e(2)*e(3)*e(4)
```

See also: Section D.14.1.29 [arrOrlikSolomon], page 2221.

D.14.1.30 arrDer

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrDer(A); arr A , multarr A

Return: [module] The module Der(A) of derivations of the (multi-)arrangement A, i.e. the derivations tangent to each hyperplane of A (resp. with multiplicities)

Note: This is only defined for central (multi-)arrangements

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A3 = arrBoolean(3);
arr B3 = arrTypeB(3);
arr G = ideal(x,y,z,x+y+z);
//The derivation module of the Boolean 3-arrangement:
arrDer(A3);
↦ _[1]=z*gen(3)
↦ _[2]=y*gen(2)
↦ _[3]=x*gen(1)
//The derivation module of the Braid 3-arrangement:
arrDer(B3);
↦ _[1]=x*gen(1)+y*gen(2)+z*gen(3)
↦ _[2]=x2y*gen(2)-y3*gen(2)+x2z*gen(3)-z3*gen(3)
↦ _[3]=x2y2z*gen(3)-x2z3*gen(3)-y2z3*gen(3)+z5*gen(3)
//The derivation module of the generic arrangement:
arrDer(G);
↦ _[1]=x*gen(1)+y*gen(2)+z*gen(3)
↦ _[2]=yz*gen(3)-yz*gen(2)
↦ _[3]=xz*gen(3)+yz*gen(2)+z2*gen(3)
↦ _[4]=xy*gen(2)+y2*gen(2)+yz*gen(2)
```

See also: Section D.14.1.30 [arrDer], page 2222; Section D.14.1.32 [arrExponents], page 2223; Section D.14.1.31 [arrIsFree], page 2223.

D.14.1.31 arrIsFree

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrIsFree(A); arr A, multarr A

Return: [0,1] 1 if the (multi-)arrangement is free, i.e. Der(A) is a free module

Note: only defined for central arrangements

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A3 = arrBoolean(3);
arr B3 = arrTypeB(3);
arr G = ideal(x,y,z,x+y+z);
arrIsFree(A3);
↦ 1
arrIsFree(B3);
↦ 1
arrIsFree(G);
↦ 0
```

See also: Section D.14.1.30 [arrDer], page 2222; Section D.14.1.32 [arrExponents], page 2223; Section D.14.1.31 [arrIsFree], page 2223.

D.14.1.32 arrExponents

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrExponents(A); arr A, multarr A

Return: [intvec] The exponents of a free (multi-) arrangement, i.e. the degrees of a basis of D(A) the derivation module.

Note: only defined for central arrangements

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A3 = arrBoolean(3);
arr B3 = arrTypeB(3);
arr G = ideal(x,y,z,x+y+z);
arrExponents(A3);
↦ 1,1,1
arrExponents(B3);
↦ 1,3,5
```

See also: Section D.14.1.30 [arrDer], page 2222; Section D.14.1.32 [arrExponents], page 2223; Section D.14.1.31 [arrIsFree], page 2223.

D.14.1.33 arr2multarr

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: multArrFromIntvec(arr A, intvec v);

Return: [multarr] multiarrangement MA, which is the arrangement A with multiplicities v

Note: the size of v must match the number of hyperplanes of the arrangement A

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
arr A = arrTypeB(3);              A;
↦ _[1]=x-y
↦ _[2]=x+y
↦ _[3]=x-z
↦ _[4]=x+z
↦ _[5]=x
↦ _[6]=y-z
↦ _[7]=y+z
↦ _[8]=y
↦ _[9]=z
↦
intvec v=2:9;               v;
↦ 2,2,2,2,2,2,2,2,2
multarr MA=arr2multarr(A,v);
MA;
↦ _[1]=(x-y)^2
↦ _[2]=(x+y)^2
↦ _[3]=(x-z)^2
↦ _[4]=(x+z)^2
↦ _[5]=(x)^2
↦ _[6]=(y-z)^2
↦ _[7]=(y+z)^2
↦ _[8]=(y)^2
↦ _[9]=(z)^2
↦
```

See also: Section D.14.1.33 [arr2multarr], page 2224; Section D.14.1.34 [multarr2arr], page 2224.

D.14.1.34 multarr2arr

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: multarr2arr(multarr A, intvec v);

Return: [arr] arrangement A, with all multiplicities removed

Example:

```
LIB "arr.lib";
ring r = 0,(x,y,z),dp;
multarr A=x2y3z5;   A;
↦ _[1]=(x)^2
↦ _[2]=(y)^3
↦ _[3]=(z)^5
↦
```

```
arr AS = multarr2arr(A); AS;
↦ _[1]=x
↦ _[2]=y
↦ _[3]=z
↦
```

See also: Section D.14.1.33 [arr2multarr], page 2224; Section D.14.1.34 [multarr2arr], page 2224.

D.14.1.35 multarrRestrict

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Return: [multarr] the restricted hyperplane Multi-Arrangement (A^X) with multiplicities i.e. counting how often one element of the restricted arrangement occurs as intersetion of hyperplane of the first arrangement. This definition is due to Guenter M. Ziegler.

Note: A has to be non-empty.

Remarks: We restrict A to the flat X, defined by the equations in A[v]. The restriction will only be performed, if the ideal defining the flat X is monomial (i.e. X is an intersection of coordinate planes). If the optional argument CC is given, the arrangement is transformed in such a way that X has the above form.

Example:

```
LIB "arr.lib";
ring R = 0,x(1..5),dp;
arr A = arrEdelmanReiner();   A;
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ _[3]=x(3)
↦ _[4]=x(4)
↦ _[5]=x(5)
↦ _[6]=x(1)-x(2)-x(3)-x(4)-x(5)
↦ _[7]=x(1)-x(2)-x(3)-x(4)+x(5)
↦ _[8]=x(1)-x(2)-x(3)+x(4)-x(5)
↦ _[9]=x(1)-x(2)-x(3)+x(4)+x(5)
↦ _[10]=x(1)-x(2)+x(3)-x(4)-x(5)
↦ _[11]=x(1)-x(2)+x(3)-x(4)+x(5)
↦ _[12]=x(1)-x(2)+x(3)+x(4)-x(5)
↦ _[13]=x(1)-x(2)+x(3)+x(4)+x(5)
↦ _[14]=x(1)+x(2)-x(3)-x(4)-x(5)
↦ _[15]=x(1)+x(2)-x(3)-x(4)+x(5)
↦ _[16]=x(1)+x(2)-x(3)+x(4)-x(5)
↦ _[17]=x(1)+x(2)-x(3)+x(4)+x(5)
↦ _[18]=x(1)+x(2)+x(3)-x(4)-x(5)
↦ _[19]=x(1)+x(2)+x(3)-x(4)+x(5)
↦ _[20]=x(1)+x(2)+x(3)+x(4)-x(5)
↦ _[21]=x(1)+x(2)+x(3)+x(4)+x(5)
↦
multarr AR = multarrRestrict(A,6,"CC");  AR;
↦ _[1]=(x(2)+1/4*x(3)+1/4*x(4)+1/4*x(5))^2
↦ _[2]=(x(2)+4*x(3)-x(4)-x(5))^2
↦ _[3]=(x(2)-x(3)+4*x(4)-x(5))^2
↦ _[4]=(x(2)-x(3)-x(4)+4*x(5))^2
↦ _[5]=(x(2)-x(3)-x(4)-x(5))^2
↦ _[6]=(x(2)-x(3)-x(4)+3/2*x(5))^1
↦ _[7]=(x(2)-x(3)+3/2*x(4)-x(5))^1
↦ _[8]=(x(2)-x(3)+3/2*x(4)+3/2*x(5))^1
↦ _[9]=(x(2)-x(3)+2/3*x(4)+2/3*x(5))^1
↦ _[10]=(x(2)+3/2*x(3)-x(4)-x(5))^1
↦ _[11]=(x(2)+3/2*x(3)-x(4)+3/2*x(5))^1
↦ _[12]=(x(2)+2/3*x(3)-x(4)+2/3*x(5))^1
↦ _[13]=(x(2)+3/2*x(3)+3/2*x(4)-x(5))^1
↦ _[14]=(x(2)+2/3*x(3)+2/3*x(4)-x(5))^1
↦ _[15]=(x(2)+2/3*x(3)+2/3*x(4)+2/3*x(5))^1
↦
```

See also: Section D.14.1.19 [arrRestrict], page 2215; Section D.14.1.36 [multarrMultRestrict], page 2226; Section D.14.1.35 [multarrRestrict], page 2225.

D.14.1.36 multarrMultRestrict

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: multarrMultRestrict(A, k); multiarrangement A, integer k;

Return: [multarr] the restricted hyperplane Multi-Arrangement (A^H_k) with multiplicities, i.e. counting with multiplicities how often one element of the restricted arrangement occurs as intersetion of hyperplane of the first multiarrangement. This definition is due to Guenter M. Ziegler.

Note: A has to be non-empty.

Remarks: The restriction will only be performed, if H_k = ker(x_i) for some i. One can also restrict an arrangement with respect to any hyper- plane k, but than a coordinate change is necessary first to make H_k = ker(x_k). Since such a coordinate change is not unique, please use arrCoordchange to do so.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z),dp;
multarr A =ideal(x2,y2,z2,(x-y)^3,(x-z)^2,(y-z));    A;
↦ _[1]=(x)^2
↦ _[2]=(y)^2
↦ _[3]=(z)^2
↦ _[4]=(x-y)^3
↦ _[5]=(x-z)^2
↦ _[6]=(y-z)^1
↦
//The restriction of the multiarrangement is:
multarr AR = multarrMultRestrict(A,1);  AR;
↦ _[1]=(y)^5
↦ _[2]=(z)^4
↦ _[3]=(y-z)^1
↦
```

See also: Section D.14.1.19 [arrRestrict], page 2215; Section D.14.1.36 [multarrMultRestrict], page 2226; Section D.14.1.35 [multarrRestrict], page 2225.

D.14.1.37 arrFlats

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: size(A); A arr

Return: [arrposet] Intersection lattice

Example:

```
LIB "arr.lib";
ring R = 0,(x(1..5)),dp;
arrFlats(arrBraid(5));
↦
↦
↦ === Computing poset ===
↦
↦
↦ rank: 2, expected OPS: 45, executed OPS: 60
↦ Cleaned up: 35 hyperplanes
↦
↦
↦ rank: 3, expected OPS: 300, executed OPS: 75
↦ Cleaned up: 60 hyperplanes
↦
↦
↦ rank: 4, expected OPS: 105, executed OPS: 15
↦ Cleaned up: 14 hyperplanes
↦
↦
↦
↦
↦ Matrix tests: 1484
↦ Given Arrangement:
↦ _[1]=x(1)-x(2)
↦ _[2]=x(1)-x(3)
↦ _[3]=x(1)-x(4)
↦ _[4]=x(1)-x(5)
↦ _[5]=x(2)-x(3)
↦ _[6]=x(2)-x(4)
↦ _[7]=x(2)-x(5)
↦ _[8]=x(3)-x(4)
↦ _[9]=x(3)-x(5)
↦ _[10]=x(4)-x(5)
↦
↦ Corresponding poset:
↦ ====== rank 1: 10 flats ======
↦  (1),  (2),  (3),  (4),  (5),  (6),  (7),  (8),  (9),  (10),
↦ ====== rank 2: 25 flats ======
↦  (1,2,5),  (1,3,6),  (1,4,7),  (1,8),  (1,9),  (1,10),  (2,3,8),  (2,4,9)\
   ,  (2,6),  (2,7),  (2,10),  (3,4,10),  (3,5),  (3,7),  (3,9),  (4,5),  (4\
   ,6),  (4,8),  (5,6,8),  (5,7,9),  (5,10),  (6,7,10),  (6,9),  (7,8),  (8,\
   9,10),
↦ ====== rank 3: 15 flats ======
↦  (1,2,3,5,6,8),  (1,2,4,5,7,9),  (1,2,5,10),  (1,3,4,6,7,10),  (1,3,6,9),\
    (1,4,7,8),  (1,8,9,10),  (2,3,4,8,9,10),  (2,3,7,8),  (2,4,6,9),  (2,6,\
   7,10),  (3,4,5,10),  (3,5,7,9),  (4,5,6,8),  (5,6,7,8,9,10),
↦ ====== rank 4: 1 flats ======
↦  (1,2,3,4,5,6,7,8,9,10),
↦ ====== rank 5: 0 flats ======
↦
↦
```

See also: Section D.14.1.37 [arrFlats], page 2226.

D.14.1.38 arrLattice

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrLattice(arr ARR)

Return: [arrposet] intersection poset of the arrangement

Note: The algorithm works by a bottom up approach, i.e. it calculates the

Example:

```
LIB "arr.lib";
ring r;
arrLattice(arrTypeB(3));
↦
↦
↦ === Computing poset ===
↦
↦
↦ rank 2: found 13 flats in 0s
↦ rank 3: found 1 flats in 0s
↦
↦
↦ Matrix tests: 86
↦ Given Arrangement:
↦ _[1]=x-y
↦ _[2]=x+y
↦ _[3]=x-z
↦ _[4]=x+z
↦ _[5]=x
↦ _[6]=y-z
↦ _[7]=y+z
↦ _[8]=y
↦ _[9]=z
↦
↦ Corresponding poset:
↦ ====== rank 1: 9 flats ======
↦  (1),  (2),  (3),  (4),  (5),  (6),  (7),  (8),  (9),
↦ ====== rank 2: 13 flats ======
↦  (1,2,5,8),  (1,3,6),  (1,4,7),  (1,9),  (2,3,7),  (2,4,6),  (2,9),  (3,4\
  ,5,9),  (3,8),  (4,8),  (5,6),  (5,7),  (6,7,8,9),
↦ ====== rank 3: 1 flats ======
↦  (1,2,3,4,5,6,7,8,9),
↦
```

See also: Section D.14.1.37 [arrFlats], page 2226; Section D.14.1.38 [arrLattice], page 2228.

D.14.1.39 moebius

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: moebius(arrposet P)

Return: [arrposet] fills in the moebius values of the flats in the poset

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z,t),dp;
arr A = arrBraid(4);
arrposet P = arrLattice(A);
↦
↦
↦ === Computing poset ===
↦
↦
↦ rank 2: found 7 flats in 0s
↦ rank 3: found 1 flats in 0s
↦
↦
↦ Matrix tests: 38
P;
↦ Given Arrangement:
↦ _[1]=x-y
↦ _[2]=x-z
↦ _[3]=x-t
↦ _[4]=y-z
↦ _[5]=y-t
↦ _[6]=z-t
↦
↦ Corresponding poset:
↦ ====== rank 1: 6 flats ======
↦  (1),  (2),  (3),  (4),  (5),  (6),
↦ ====== rank 2: 7 flats ======
↦  (1,2,4),  (1,3,5),  (1,6),  (2,3,6),  (2,5),  (3,4),  (4,5,6),
↦ ====== rank 3: 1 flats ======
↦  (1,2,3,4,5,6),
↦ ====== rank 4: 0 flats ======
↦
↦
//As you can see the values are not calculated yet:
printMoebius(P);
↦ Moebius values:
↦ ====== rank 1: 6 flats ======
↦  (-1),  (-1),  (-1),  (-1),  (-1),  (-1),
↦ ====== rank 2: 7 flats ======
↦  (0),  (0),  (0),  (0),  (0),  (0),  (0),
↦ ====== rank 3: 1 flats ======
↦  (0),
↦ ====== rank 4: 0 flats ======
↦
P = moebius(P);
//Now all entries are initialized:
printMoebius(P);
↦ Moebius values:
↦ ====== rank 1: 6 flats ======
↦  (-1),  (-1),  (-1),  (-1),  (-1),  (-1),
↦ ====== rank 2: 7 flats ======
↦  (2),  (2),  (1),  (2),  (1),  (1),  (2),
↦ ====== rank 3: 1 flats ======
↦  (-6),
↦ ====== rank 4: 0 flats ======
↦
```

See also: Section D.14.1.39 [moebius], page 2228.

D.14.1.40 arrCharPoly

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrCharPoly(arr A)

Return: [intvec] coefficients of the characteristic polynomial of A in increasing order

Remarks: The algorithm only returns the coefficients of the characteristic polynomial since they are whole numbers but the basering could be something different.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z,u,v),dp;
arr A = arrBraid(5);
intvec v = arrCharPoly(A);
x*(x-1)*(x-2)*(x-3)*(x-4);
↦ x5-10x4+35x3-50x2+24x
v;
↦ 24,-50,35,-10,1
```

See also: Section D.14.1.40 [arrCharPoly], page 2230; Section D.14.1.41 [arrPoincare], page 2230.

D.14.1.41 arrPoincare

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrPoincare(A); arr A

Return: [intvec] The Poincare polynomial as integer vector of the arrangement, which is equal to the second kind Poincare-Series of the Orlik-Solomon Algebra.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z,u,v),dp;
arr A = arrBraid(5);
intvec v = arrPoincare(A);
(1+x)*(1+2x)*(1+3x)*(1+4x);
↦ 24x4+50x3+35x2+10x+1
v;
↦ 1,10,35,50,24
```

See also: Section D.14.1.43 [arrBoundedChambers], page 2231; Section D.14.1.42 [arrChambers], page 2231; Section D.14.1.41 [arrPoincare], page 2230.

D.14.1.42 arrChambers

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrChambers(A); arr A

Return: [int] The number of chambers of an arrangement, which is equal to the evaluation of the Poincare polynomial at 1.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y),dp;
arr A = ideal(x,y,x+y-1);
arrChambers(A);
↦ 7
```

See also: Section D.14.1.43 [arrBoundedChambers], page 2231; Section D.14.1.42 [arrChambers], page 2231; Section D.14.1.41 [arrPoincare], page 2230.

D.14.1.43 arrBoundedChambers

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: arrBoundedChambers(A); arr A

Return: [int] The number of bounded chambers of an arrangement, which is equal to the evaluation of the Poincare polynomial at -1.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y),dp;
arr A = ideal(x,y,x+y-1);
arrBoundedChambers(A);
↦ 1
```

See also: Section D.14.1.43 [arrBoundedChambers], page 2231; Section D.14.1.42 [arrChambers], page 2231; Section D.14.1.41 [arrPoincare], page 2230.

D.14.1.44 printMoebius

Procedure from library `arr.lib` (see Section D.14.1 [arr_lib], page 2205).

Usage: printMoebius(A); arr A

Return: [] displays the moebius values of all the flats in the poset

Remarks: Mainly used for debugging.

Example:

```
LIB "arr.lib";
ring R = 0,(x,y,z,t),dp;
arr A = arrBraid(4);
arrposet P = arrLattice(A);
↦
↦
↦ === Computing poset ===
↦
↦
↦ rank 2: found 7 flats in 0s
↦ rank 3: found 1 flats in 0s
↦
↦
↦ Matrix tests: 38
P;
↦ Given Arrangement:
↦ _[1]=x-y
↦ _[2]=x-z
↦ _[3]=x-t
↦ _[4]=y-z
↦ _[5]=y-t
↦ _[6]=z-t
↦
↦ Corresponding poset:
↦ ====== rank 1: 6 flats ======
↦  (1),  (2),  (3),  (4),  (5),  (6),
↦ ====== rank 2: 7 flats ======
↦  (1,2,4),  (1,3,5),  (1,6),  (2,3,6),  (2,5),  (3,4),  (4,5,6),
↦ ====== rank 3: 1 flats ======
↦  (1,2,3,4,5,6),
↦ ====== rank 4: 0 flats ======
↦
↦
//As you can see the values are not calculated yet:
printMoebius(P);
↦ Moebius values:
↦ ====== rank 1: 6 flats ======
↦  (-1),  (-1),  (-1),  (-1),  (-1),  (-1),
↦ ====== rank 2: 7 flats ======
↦  (0),  (0),  (0),  (0),  (0),  (0),  (0),
↦ ====== rank 3: 1 flats ======
↦  (0),
↦ ====== rank 4: 0 flats ======
↦
P = moebius(P);
//Now all entries are initialized:
printMoebius(P);
↦ Moebius values:
↦ ====== rank 1: 6 flats ======
↦  (-1),  (-1),  (-1),  (-1),  (-1),  (-1),
↦ ====== rank 2: 7 flats ======
↦  (2),  (2),  (1),  (2),  (1),  (1),  (2),
↦ ====== rank 3: 1 flats ======
↦  (-6),
↦ ====== rank 4: 0 flats ======
↦
```

D.14.2 combinat_lib

Library: combinat.lib

Purpose: Some useful functions

Authors: J. Boehm, boehm @ mathematik.uni-kl.de

Overview: Some useful basic functions from combinatorics.

Procedures:

D.14.2.1 intersectLists

Procedure from library `combinat.lib` (see Section D.14.2 [combinat_lib], page 2232).

Usage: intersectLists(L,M); L list, M list

Return: list, containing all elements of L which are elements of M

Theory: we require the elements of the lists to have a compare operator ==.

Example:

```
LIB "combinat.lib";
intersectLists(list(1,1,2,2,3),list(2,3,3,4));
↦ [1]:
↦    2
↦ [2]:
↦    3
intersectLists(list(1,1,2,2,3),list(1,1,2,2,3));
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ [3]:
↦    3
```

D.14.2.2 sublists

Procedure from library `combinat.lib` (see Section D.14.2 [combinat_lib], page 2232).

Usage: sublists(L); L list

Assume: L is a list

Return: list of all sublists of L.

Example:

```
LIB "combinat.lib";
list L = 1,2,3,4,5;
sublists(L);
↦ [1]:
↦    empty list
↦ [2]:
↦    [1]:
↦       1
↦ [3]:
↦    [1]:
↦       2
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦ [5]:
↦    [1]:
↦       3
↦ [6]:
↦    [1]:
↦       1
↦    [2]:
↦       3
↦ [7]:
↦    [1]:
↦       2
↦    [2]:
↦       3
↦ [8]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦    [3]:
↦       3
↦ [9]:
↦    [1]:
↦       4
↦ [10]:
↦    [1]:
↦       1
↦    [2]:
↦       4
↦ [11]:
↦    [1]:
↦       2
↦    [2]:
↦       4
↦ [12]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦    [3]:
↦       4
↦ [13]:
↦    [1]:
↦       3
↦    [2]:
↦       4
↦ [14]:
↦    [1]:
↦       1
↦    [2]:
↦       3
↦    [3]:
```

```
↦         4
↦  [15]:
↦     [1]:
↦         2
↦     [2]:
↦         3
↦     [3]:
↦         4
↦  [16]:
↦     [1]:
↦         1
↦     [2]:
↦         2
↦     [3]:
↦         3
↦     [4]:
↦         4
↦  [17]:
↦     [1]:
↦         5
↦  [18]:
↦     [1]:
↦         1
↦     [2]:
↦         5
↦  [19]:
↦     [1]:
↦         2
↦     [2]:
↦         5
↦  [20]:
↦     [1]:
↦         1
↦     [2]:
↦         2
↦     [3]:
↦         5
↦  [21]:
↦     [1]:
↦         3
↦     [2]:
↦         5
↦  [22]:
↦     [1]:
↦         1
↦     [2]:
↦         3
↦     [3]:
↦         5
↦  [23]:
↦     [1]:
↦         2
↦     [2]:
```

```
↦          3
↦       [3]:
↦          5
↦    [24]:
↦       [1]:
↦          1
↦       [2]:
↦          2
↦       [3]:
↦          3
↦       [4]:
↦          5
↦    [25]:
↦       [1]:
↦          4
↦       [2]:
↦          5
↦    [26]:
↦       [1]:
↦          1
↦       [2]:
↦          4
↦       [3]:
↦          5
↦    [27]:
↦       [1]:
↦          2
↦       [2]:
↦          4
↦       [3]:
↦          5
↦    [28]:
↦       [1]:
↦          1
↦       [2]:
↦          2
↦       [3]:
↦          4
↦       [4]:
↦          5
↦    [29]:
↦       [1]:
↦          3
↦       [2]:
↦          4
↦       [3]:
↦          5
↦    [30]:
↦       [1]:
↦          1
↦       [2]:
↦          3
↦       [3]:
```

```
↦       4
↦    [4]:
↦       5
↦ [31]:
↦    [1]:
↦       2
↦    [2]:
↦       3
↦    [3]:
↦       4
↦    [4]:
↦       5
↦ [32]:
↦    [1]:
↦       1
↦    [2]:
↦       2
↦    [3]:
↦       3
↦    [4]:
↦       4
↦    [5]:
↦       5
```

D.14.2.3 member

Procedure from library `combinat.lib` (see Section D.14.2 [combinat_lib], page 2232).

Usage: member(e,L); e def, L list

Return: 1 if e is an element of L, 0 otherwise

Theory: we require the elements involved to have a compare operator ==.

Example:

```
LIB "combinat.lib";
member(1,list(1,4,5));
↦ 1
```

D.14.3 customstd_lib

Library: customstd.lib

Purpose: Load customstd.so

Authors: Hans Schoenemann, hannes at mathematik.uni-kl.de
Yue Ren, ren at mathematik.uni-kl.de

Overview: This library offers customly modified standard bases algorithms in order to increase the performance of other algorithms. If you require a customly modified standard bases algorithm, please contact the authors.

Procedures:

D.14.3.1 monomialabortstd

Procedure from library `customstd.lib` (see Section D.14.3 [customstd_lib], page 2237).

Usage: monomialabortstd(I); I ideal

Purpose: computes a standard basis and aborts if a monomial generator is found. Returns all standard basis elements that have been computed.

Note: Due to sequencing of the standard basis computation:
- If aborted, there is no guarantee that the monomial is the final standard bases element
- If ideal has a monomial generator, there is no guarantee that the computation aborts because of it

Example:

```
LIB "customstd.lib";
LIB "polylib.lib";
ring r = 0,(x,y,z,u,v),dp;
// yields normal standard basis since no monomial is found
ideal I = homog(cyclic(4),v);
monomialabortstd(I);
↦ _[1]=x+y+z+u
↦ _[2]=y2+2yu+u2
↦ _[3]=yz2+z2u-yu2-u3
↦ _[4]=yzu2+z2u2-yu3+zu3-u4-v4
↦ _[5]=yu4+u5-yv4-uv4
↦ _[6]=z3u2+z2u3-zv4-uv4
↦ _[7]=z2u4+yzv4-yuv4+zuv4-2u2v4
// will not start standard basis computation in the first place,
// as one of the generators is a monomial
I = v,homog(cyclic(4),v);
monomialabortstd(I);
↦ _[1]=v
// aborts standard basis computation when it encounters monomial u
// note that u is not the final element in the standard basis!!!
I = x+y+z,homog(cyclic(4),v);
monomialabortstd(I);
↦ _[1]=u
↦ _[2]=x+y+z+u
// aborts standard basis computation when it encounters monomial z2
// note that it is not the monomial generator x which lead to the abortion!!!
I = x,homog(cyclic(4),v);
monomialabortstd(I);
↦ _[1]=y+z+u
↦ _[2]=x+y+z+u
↦ _[3]=z2
```

D.14.3.2 satstd

Procedure from library `customstd.lib` (see Section D.14.3 [customstd_lib], page 2237).

Usage: satstd(I[,J]); I ideal, J optional ideal

Assume: J generated by variables

Purpose: computes a standard basis of I and, if possible,
divides each polynomial during the computation by the variables in J. By default, J is assumed to be the ideal generated by all variables.

Note: Even if I contains a monomial generated by the variables in J, there is no guarantee that it is found during the computation. If it is found, however, 1 is immediately returned.

Note: The result is a standard basis of a partially saturated ideal wrt. the the variables in J. If the I is homogeneous and the ordering dp, the result is completely saturated wrt. to the last variable (wrt. to the first for Dp).

Example:

```
LIB "customstd.lib";
ring r = 0,(x,y,z,u,v),dp;
ideal I = x2+x,y2+y,z2+z;
// returns normal standard basis, no changes during (trivial) computation
satstd(I,ideal(u,v));
↦ _[1]=z2+z
↦ _[2]=y2+y
↦ _[3]=x2+x
// returns x+1 instead of x2+x
satstd(I,ideal(x));
↦ _[1]=x+1
↦ _[2]=z2+z
↦ _[3]=y2+y
// returns standard basis with elements of degree up to 8 (instead of 16)
deg(satstd(I^8));
↦ 8
```

D.14.4 methods_lib

Library: methods.lib

Purpose: installing methods in Singular

Authors: J. Boehm, boehm @ mathematik.uni-kl.de

Overview: Methods select the function to execute by the types of the input tuple. The central function is installMethod, which takes a hashtable associating a tuple of input types to function names and creates a corresponding procedure.

HashTables are lists with arbitrary index sets. They can be created by the command hashTable. Their size can be determined by the command size. Values can be extracted by selectKey or the * operator. HashTables can also be added using addHashTables or the + operator.

Methods can be added with the + operator.

Types: Method the class of all methods
HashTable the class of all hash tables

Procedures:

D.14.5 nets_lib

Library: net.lib

Purpose: Net structures for pretty printing

Authors: J. Boehm, boehm@mathematik.uni-kl.de
M. Mueller, mkmuelle@mathematik.uni-kl.de
H. Rombach, rombach@mathematik.uni-kl.de
M. Stein, maxstein77@web.de

Overview: Nets are arrays of characters, which are printed in a matrix format. They can be concatenated horizontally and vertically. When concatenating horizontally, empty rows are filled with spaces. All Singular types can be converted to a Net by applying the command net.

Types: Net The class of all nets

Procedures:

D.14.5.1 catNets

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: catNets(N,M); N,M Nets

Return: horizontal concatenation of N and M

Theory: The nets are concated horizontally with alignment at the top row. Can also be called with `+`.

Example:

```
LIB "nets.lib";
Net A = net("aaa");
Net B = net("b");
catNets(A,B);
↦ aaab
↦
A+B;
↦ aaab
↦
```

D.14.5.2 net

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

D.14.5.3 netBigIntMat

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netBigIntMat(M); M bigint Matrix

Assume: M is a bigintmatrix

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
int a=111111;
int b=22222;
int c=3333;
int d=444;
bigintmat M[2][2]=a,b,c,d;
netBigIntMat(M);
↦ | 111111 22222 |
↦ | 3333   444   |
↦
```

D.14.5.4 netBigIntMatShort

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netBigIntMatShort(M); M bigint matrix

Assume: M is a bigintmatrix

Return: visual presentation of M, only the first digits of each entry

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
bigint a=1111111111;
bigint b=2222222222;
bigint c=3333333333;
bigint d=4444444444;
bigintmat M[2][2]=a,b,c,d;
netBigIntMatShort(M);
↦ | 1111111111 2222222222 |
↦ | 3333333333 4444444444 |
↦
```

D.14.5.5 netCoefficientRing

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netCoefficientRing(R); R ring

Assume: R is a ring

Return: visual presentation of the coefficient ring of R

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r=0,x,lp;
netCoefficientRing(r);
↦ QQ
↦
```

D.14.5.6 netIdeal

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netIdeal(P); P ideal

Assume: P is a poly

Return: visual presentation of I

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r;
ideal I=2x3y4,2x3y4z+x3y4z5,5x6y7z10-2x3y+4y5z8;
netIdeal(I);
↦ <2x3y4, x3y4z5+2x3y4z, 5x6y7z10+4y5z8-2x3y>
↦
```

D.14.5.7 netInt

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netInt(M); M integer

Assume: M is an integer

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r;
int M=5;
netInt(M);
↦ 5
↦
```

D.14.5.8 netBigInt

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netBigInt(M); M integer

Assume: M is a bigint

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r;
bigint M=5;
netBigInt(M);
↦ 5
↦
```

D.14.5.9 netIntMat

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netIntMat(M); M int matrix

Assume: M is a int matrix

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
int a=111111;
int b=222222;
int c=333333;
int d=444444;
intmat M[2][2]=a,b,c,d;
netIntMat(M);
↦ | 111111 222222 |
↦ | 333333 444444 |
↦
```

D.14.5.10 netIntMatShort

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netIntMatShort(M); M int matrix

Assume: M is a int matrix

Return: visual presentation of M, only the first digits of each entry

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
int a=111111;
int b=222222;
int c=333333;
int d=444444;
intmat M[2][2]=a,b,c,d;
print(M);
↦  111111 222222
↦  333333 444444
netIntMatShort(M);
↦ | 111111 222222 |
↦ | 333333 444444 |
↦
//
print(M);
↦  111111 222222
↦  333333 444444
```

```
netIntMatShort(M,2);
↦ | 11... 22... |
↦ | 33... 44... |
↦
//
```

D.14.5.11 netIntVector

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netIntVector(V); V int vector

Assume: V is a int vector

Return: visual presentation of V

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
int a=111111;
int b=222222;
int c=333333;
int d=444444;
intvec V=a,b,c,d;
netIntVector(V);
↦ | 111111 |
↦ | 222222 |
↦ | 333333 |
↦ | 444444 |
↦
```

D.14.5.12 netIntVectorShort

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netIntVectorShort(V); V int vector

Assume: V is a int vector

Return: visual presentation of V, only the first digits of each entry

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
int a=111111;
int b=222222;
int c=333333;
int d=444444;
intvec V=a,b,c,d;
netIntVectorShort(V,4);
↦ | 1111... |
↦ | 2222... |
↦ | 3333... |
↦ | 4444... |
↦
```

D.14.5.13 netNumber

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netNumber(R); a number

Assume: a is a number

Return: visual presentation of a

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
number a = 5;
netNumber(a);
↦
↦ 5
↦
```

D.14.5.14 netList

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netList(L); L list

Assume: L is a list of elements

Return: visual presentation of L

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
list L=111, x+y, x3y-z, y+3z4;
print("list L=111, x+y, x3y-z, y+3z4");
↦ list L=111, x+y, x3y-z, y+3z4
netList(L);
↦ [111, x+y, x3y-z, y+3z4]
↦
```

D.14.5.15 netMap

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netMap(f); f map

Assume: f is a map from a ring to the basering

Return: visual presentation of the map f

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
ring r2=103,(a,b,c),lp;
map f=r1,ab,ac,bc;
netMap(f);
↦ Map: r1 --> r2 , x -> ab
↦                , y -> ac
↦                , z -> bc
↦
```

D.14.5.16 netMap2

Procedure from library nets.lib (see Section D.14.5 [nets_lib], page 2239).

Usage: netMap2(f); f map

Assume: f is a map from a ring to the basering

Return: visual presentation of the map f, alternative version

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
ring r2=103,(a,b,c),lp;
map f=r1,ab,ac,bc;
netMap2(f);
↦ f: r1 --> r2 , x -> ab , y -> ac , z -> bc
↦
```

D.14.5.17 netmatrix

Procedure from library nets.lib (see Section D.14.5 [nets_lib], page 2239).

Usage: netmatrix(V); M matrix

Assume: M is a matrix

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
poly a=2x3y4+300xy-234z23;
poly b=2x3y4z;
poly c=x3y4z5;
poly d=5x6y7z10;
poly e=2x3y;
poly f=4y5z8;
matrix M[2][3]=a,b,c,d,e,f;
print(M);
↦ 2x3y4-3xy-32z23,2x3y4z,x3y4z5,
↦ 5x6y7z10,       2x3y,  4y5z8
netmatrix(M);
↦ | 2x3y4-3xy-32z23 2x3y4z x3y4z5 |
↦ | 5x6y7z10        2x3y   4y5z8  |
↦
```

D.14.5.18 netmatrixShort

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netmatrixShort(M); M matrix

Assume: M is a matrix

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
poly a=2x3y4+300xy-234z23;
poly b=2x3y4z;
poly c=x3y4z5;
poly d=5x6y7z10;
poly e=2x3y-2x3y4+300xy-234z23;
poly f=4y5z8;
matrix M[2][3]=a,b,c,d,e,f;
netmatrixShort(M, 10);
↦ | 2x3y4-3xy-32z23 2x3y4z                x3y4z5 |
↦ | 5x6y7z10        -2x3y4+2x3y-3xy-32z23 4y5z8  |
↦
```

D.14.5.19 netPoly

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netPoly(P); P poly

Assume: P is a poly

Return: visual presentation of P over two rows

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
// from 3.3.1 Examples of ring declarations
//
ring R1 = 32003,(x,y,z),dp;
poly q6=1;
```

```
print(q6);
↦ 1
netPoly(q6);
↦
↦ 1
↦
poly q7=-1;
print(q7);
↦ -1
netPoly(q7);
↦
↦ -1
↦
poly q8=2;
print(q8);
↦ 2
netPoly(q8);
↦
↦ 2
↦
poly q9=-2;
print(q9);
↦ -2
netPoly(q9);
↦
↦ -2
↦
poly q1=x+y+z;
print(q1);
↦ x+y+z
netPoly(q1);
↦
↦ x+y+z
↦
poly q2=xy+xz+yz;
print(q2);
↦ xy+xz+yz
netPoly(q2);
↦
↦ xy+xz+yz
↦
poly q3=2x3y3z4-3x4y5z6;
print(q3);
↦ -3x4y5z6+2x3y3z4
netPoly(q3);
↦    4 5 6   3 3 4
↦ -3x y z +2x y z
↦
poly q4=x3y3z4-x4y5z6;
print(q4);
↦ -x4y5z6+x3y3z4
netPoly(q4);
↦   4 5 6  3 3 4
↦ -x y z +x y z
↦
poly q5=-x3y3z4+x4y5z6;
print(q5);
↦ x4y5z6-x3y3z4
netPoly(q5);
↦  4 5 6  3 3 4
↦ x y z -x y z
↦
ring R2 = 32003,(x(1..10)),dp;
poly w6=1;
print(w6);
↦ 1
netPoly(w6);
↦
↦ 1
↦
poly w7=-1;
print(w7);
↦ -1
netPoly(w7);
↦
↦ -1
↦
poly w2=-x(1)-(2)-x(3);
print(w2);
↦ -x(1)-x(3)-2
netPoly(w2);
↦
↦ -x(1)-x(3)-2
↦
poly w3=x(1)*x(2)+x(1)*x(2)+x(2)*x(3);
print(w3);
↦ 2*x(1)*x(2)+x(2)*x(3)
netPoly(w3);
↦
↦ 2x(1)x(2)+x(2)x(3)
↦
poly w4=x(1)*x(2)-x(1)*x(2)-x(2)*x(3);
print(w4);
↦ -x(2)*x(3)
netPoly(w4);
↦
↦ -x(2)x(3)
↦
poly w5=x(1)^2*x(2)^3*x(3)^4;
print(w5);
↦ x(1)^2*x(2)^3*x(3)^4
netPoly(w5);
↦     2    3    4
↦ x(1) x(2) x(3)
↦
poly w8=x(1)+x(2)+x(3);
```

```
print(w8);
↦ x(1)+x(2)+x(3)
netPoly(w8);
↦ 
↦ x(1)+x(2)+x(3)
↦ 
poly w9=x(1)+x(2)+x(3);
print(w9);
↦ x(1)+x(2)+x(3)
netPoly(w9);
↦ 
↦ x(1)+x(2)+x(3)
↦ 
ring R3 = 32003,(x(1..5)(1..8)),dp;
poly e1=x(1)(1)+x(2)(2)+x(3)(3);
print(e1);
↦ x(1)(1)+x(2)(2)+x(3)(3)
netPoly(e1);
↦ 
↦ x(1)(1)+x(2)(2)+x(3)(3)
↦ 
poly e2=x(1)(1)*x(2)(2)*x(3)(3);
print(e2);
↦ x(1)(1)*x(2)(2)*x(3)(3)
netPoly(e2);
↦ 
↦ x(1)(1)x(2)(2)x(3)(3)
↦ 
poly e3=x(1)(1)^2*x(2)(2)^3*x(3)(3)^4;
print(e3);
↦ x(1)(1)^2*x(2)(2)^3*x(3)(3)^4
netPoly(e3);
↦        2       3       4
↦ x(1)(1) x(2)(2) x(3)(3)
↦ 
poly e4=-x(1)(1)^2*x(2)(2)^3*x(3)(3)^4-x(1)(1)^3*x(2)(2)^3*x(3)(3)^4;
print(e4);
↦ -x(1)(1)^3*x(2)(2)^3*x(3)(3)^4-x(1)(1)^2*x(2)(2)^3*x(3)(3)^4
netPoly(e4);
↦         3       3       4        2       3       4
↦ -x(1)(1) x(2)(2) x(3)(3) -x(1)(1) x(2)(2) x(3)(3)
↦ 
ring r=32003,(x,y,z),lp;
poly p=x4+4y4+4z4-x3-3y3-3z3+1x2+2y2+z2-x-1y-z1;
p;
↦ x4-x3+x2-x+4y4-3y3+2y2-y+4z4-3z3+z2-z
netPoly(p);
↦  4  3  2     4   3   2     4   3  2
↦ x -x +x -x+4y -3y +2y -y+4z -3z +z -z
↦ 
poly p2=x3yz+xy3z+xyz3-2x2yz-2xy2z-2xyz2+1xyz+x1yzxy1z;
p2;
↦ x3yz+x2y2z2-2x2yz+xy3z-2xy2z+xyz3-2xyz2+xyz
```

```
netPoly(p2);
↦  3    2 2 2   2     3     2     3     2
↦ x yz+x y z -2x yz+xy z-2xy z+xyz -2xyz +xyz
↦
poly p3=x+y+z-x2-3y-4z4+xy+xz+2xy-x2y-xz2-y2z2;
p3;
↦ -x2y-x2+3xy-xz2+xz+x-y2z2-2y-4z4+z
netPoly(p3);
↦   2   2       2       2 2      4
↦ -x y-x +3xy-xz +xz+x-y z -2y-4z +z
↦
ring r2=32003,(x(1..10)),lp;
poly p=x(1)*x(2)*x(3)+2*x(1)^2+2*x(1)*x(2);
p;
↦ 2*x(1)^2+x(1)*x(2)*x(3)+2*x(1)*x(2)
netPoly(p);
↦      2
↦ 2x(1) +x(1)x(2)x(3)+2x(1)x(2)
↦
poly p2=x(1)^2*x(2)^3*x(3)^4-2*x(1)^1*x(2)^2+2*x(1)*x(2)*x(10);
p2;
↦ x(1)^2*x(2)^3*x(3)^4-2*x(1)*x(2)^2+2*x(1)*x(2)*x(10)
netPoly(p2);
↦     2    3    4          2
↦ x(1) x(2) x(3) -2x(1)x(2) +2x(1)x(2)x(10)
↦
ring r3=7,(x,y,z),lp;
poly p=17x2+24y;
p;
↦ 3x2+3y
netPoly(p);
↦   2
↦ 3x +3y
↦
ring r4=(7,a,b,c),(x,y,z),Dp;
poly p=2ax2+by-cz3;
p;
↦ (-c)*z3+(2a)*x2+(b)*y
netPoly(p);
↦      3      2
↦ (-c)z +(2a)x +(b)y
↦
ring r5=(7,a),(x,y,z),dp;
minpoly = a^2+a+3;
poly p=2ax2+y-az3;
p;
↦ (-a)*z3+(2a)*x2+y
netPoly(p);
↦      3      2
↦ (-a)z +(2a)x +y
↦
ring r6 = (complex,30,j),(x,y,z),dp;
poly p=2x2+y-z3+20*j;
```

```
p;
↦ -z3+2*x2+y+(j*20)
netPoly(p);
↦   3   2
↦ -z +2x +y+(j*20)
↦
```

D.14.5.20 netPrimePower

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netPrimePower(n,m); n,m int

Return: visual presentation of the prime power n^m

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r=0,x,lp;
int n=2;
int m=5;
netPrimePower(n,m);
↦ 2^5
↦
```

D.14.5.21 netRing

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netRing(f); f ring

Assume: R is a ring

Return: visual presentation of R

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
// from 3.3.1 Examples of ring declarations
ring r1 = 32003,(x,y,z),dp;
netRing(r1);
↦ FF_32003[x,y,z]
↦
//
ring r2 = 32003,(x(1..10)),dp;
netRing(r2);
↦ FF_32003[x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8),x(9),x(10)]
↦
//
ring r3 = 32003,(x(1..5)(1..8)),dp;
netRing(r3);
↦ FF_32003[x(1)(1),x(1)(2),x(1)(3),x(1)(4),x(1)(5),x(1)(6),x(1)(7),x(1)(8),\
   x(2)(1),x(2)(2),x(2)(3),x(2)(4),x(2)(5),x(2)(6),x(2)(7),x(2)(8),x(3)(1),x\
   (3)(2),x(3)(3),x(3)(4),x(3)(5),x(3)(6),x(3)(7),x(3)(8),x(4)(1),x(4)(2),x(\
   4)(3),x(4)(4),x(4)(5),x(4)(6),x(4)(7),x(4)(8),x(5)(1),x(5)(2),x(5)(3),x(5\
   )(4),x(5)(5),x(5)(6),x(5)(7),x(5)(8)]
↦
//
ring r4 = 0,(a,b,c,d),lp;
netRing(r4);
↦ QQ[a,b,c,d]
↦
//
ring r5 = 7,(x,y,z),ds;
netRing(r5);
↦ FF_7[x,y,z]
↦
//
ring r6 = 10,(x,y,z),ds;
↦ // ** 10 is invalid as characteristic of the ground field. 32003 is used.
netRing(r6);
↦ FF_32003[x,y,z]
↦
//
ring r7 = 7,(x(1..6)),(lp(3),dp);
netRing(r7);
↦ FF_7[x(1),x(2),x(3),x(4),x(5),x(6)]
↦
//
ring r8 = 0,(x,y,z,a,b,c),(ds(3), dp(3));
netRing(r8);
↦ QQ[x,y,z,a,b,c]
↦
//
ring r9 = 0,(x,y,z),(c,wp(2,1,3));
netRing(r9);
↦ QQ[x,y,z]
↦
//
ring r10 = (7,a,b,c),(x,y,z),Dp;
netRing(r10);
↦ FF_7(a,b,c)[x,y,z]
↦
//
ring r11 = (7,a),(x,y,z),dp;
minpoly = a^2+a+3;
netRing(r11);
↦ FF_7[a]/(a2+a+3)[x,y,z]
↦
//
ring r12 = (7^2,a),(x,y,z),dp;
netRing(r12);
↦ FF_7^2[x,y,z]
↦
//
ring r13 = real,(x,y,z),dp;
```

```
netRing(r13);
↦ QQ(6,6)[x,y,z]
↦
//
ring r14 = (real,50),(x,y,z),dp;
netRing(r14);
↦ QQ(50,50)[x,y,z]
↦
//
ring r15 = (real,10,50),(x,y,z),dp;
netRing(r15);
↦ QQ(10,50)[x,y,z]
↦
//
ring r16 = (complex,30,j),(x,y,z),dp;
netRing(r16);
↦ QQ(30,30)[x,y,z]
↦
//
ring r17 = complex,(x,y,z),dp;
netRing(r17);
↦ QQ(6,6)[x,y,z]
↦
//
ring R = 7,(x,y,z), dp;
qring r18 = std(maxideal(2));
netRing(r18);
↦ FF_7[x,y,z] / <z2, yz, xz, y2, xy, x2>
↦
```

D.14.5.22 netString

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netString(M); M string

Assume: M is a string

Return: visual presentation of M

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
string M="Hallo";
netString(M);
↦ Hallo
↦
```

D.14.5.23 netvector

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: netvector(V); V vector

Assume: V is a vector

Return: visual presentation of V

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
poly a=2x3y4;
poly b=2x3y4z;
poly c=x3y4z5;
poly d=5x6y7z10;
poly e=2x3y;
poly f=4y5z8;
vector V=[a,b,c,d,e,f];
netvector(V);
↦ | 2x3y4    |
↦ | 2x3y4z   |
↦ | x3y4z5   |
↦ | 5x6y7z10 |
↦ | 2x3y     |
↦ | 4y5z8    |
↦
```

D.14.5.24 netvectorShort

Procedure from library nets.lib (see Section D.14.5 [nets_lib], page 2239).

Usage: netvectorShort(V); V vector

Assume: V is a vector

Return: visual presentation of V, short version

Theory: A Singular object is converted into a character array (a Net) for on screen printing.

Example:

```
LIB "nets.lib";
ring r1=101,(x,y,z),lp;
poly a=2x3y4;
poly b=2x3y4z-5x6y7z10;
poly c=x3y4z5;
poly d=5x6y7z10;
poly e=2x3y;
poly f=4y5z8+5x6y7z10;
vector V=[a,b,c,d,e,f];
netvectorShort(V);
↦ | 2x3y4             |
↦ | -5x6y7z10+2x3y4z  |
↦ | x3y4z5            |
↦ | 5x6y7z10          |
↦ | 2x3y              |
↦ | 5x6y7z10+4y5z8    |
↦
```

D.14.5.25 stackNets

Procedure from library `nets.lib` (see Section D.14.5 [nets_lib], page 2239).

Usage: stackNets(N,M); N,M Nets

Return: vertical concatenation of N and M

Theory: The nets are concated vertically with alignment at the left column.

Example:

```
LIB "nets.lib";
Net A = net("aaa");
Net B = net("b");
stackNets(A,B);
↦ aaa
↦ b
↦
A+B;
↦ aaab
↦
```

D.14.6 phindex_lib

Library : phindex.lib

Purpose: Procedures to compute the index of real analytic vector fields

Author: Victor Castellanos

Note: To compute the Poincare-Hopf index of a real analytic vector field with an algebraically isolated singularity at 0 (w. an a. i. s), we use the algebraic formula for the degree of the real analytic map germ found by Eisenbud-Levine in 1997. This result was also proved by Khimshiashvili. If the isolated singularity is non algebraically isolated and the vector field has similar reduced complex zeroes of codimension 1, we use a formula as the Eisenbud-Levine found by Victor Castellanos, in both cases is necessary to use a local order (ds,...). To compute the signature of a quadratic form (or symmetric matrix) we use the method of Lagrange.

Procedures:

D.14.6.1 signatureL

Procedure from library `phindex.lib` (see Section D.14.6 [phindex_lib], page 2256).

Usage: signatureL(M[,r]); M symmetric matrix, r int (optional).

Return: the signature of M of type int or if r is given and !=0 then intvec with (signature, nr. of +, nr. of -) is returned.

Theory: Given the matrix M, we construct the quadratic form associated. Afterwards we use the method of Lagrange to compute the signature. The law of inertia for a real quadratic form A(x,x) says that in a representation of A(x,x) as a sum of independent squares A(x,x)=sum_{i=1}^r a_iX_i^2.
The number of positive and the number of negative squares are independent of the choice of representation. The signature -s- of A(x,x) is the difference between the

number -pi- of positive squares and the number -nu- of negative squares in the representation of A(x,x). The rank -r- of M (or A(x,x)) and the signature -s- determine the numbers -pi- and -nu- uniquely, since
r=pi+nu, s=pi-nu.
The method of Lagrange is a procedure to reduce any real quadratic form to a sum of squares.
Ref. Gantmacher, The theory of matrices, Vol. I, Chelsea Publishing Company, NY 1960, page 299.

Example:

```
LIB "phindex.lib";
ring r=0,(x),ds;
matrix M[5][5]=0,0,0,1,0,0,1,0,0,-1,0,0,1,0,0,1,0,0,3,0,0,-1,0,0,1;
signatureL(M,1); //The rank of M is 3+1=4
↦ 2,3,1
matrix H[5][5]=0,-7,0,1,0,-7,1,0,0,-1,0,0,1,0,0,1,0,0,-3,5,0,-1,0,5,1;
signatureL(H);
↦ 1
```

D.14.6.2 signatureLqf

Procedure from library `phindex.lib` (see Section D.14.6 [phindex_lib], page 2256).

Usage: signatureLqf(h); h quadratic form (poly type).

Return: the signature of h of type int or if r is given and !=0 then intvec with (signature, nr. of +, nr. of -) is returned.

Theory: To compute the signature we use the method of Lagrange. The law of inertia for a real quadratic form h(x,x) says that in a representation of h(x,x) as a sum of independent squares h(x,x)=sum_{i=1}^r a_i*X_i^2 the number of positive and the number of negative squares are independent of the choice of representation. The signature -s- of h(x,x) is the difference between the number -pi- of positive squares and the number -nu- of negative squares in the representation of h(x,x). The rank -r- of h(x,x) and the signature -s- determine the numbers -pi- and -nu- uniquely, since
r=pi+nu, s=pi-nu.
The method of Lagrange is a procedure to reduce any real quadratic form to a sum of squares.
Ref. Gantmacher, The theory of matrices, Vol. I, Chelsea Publishing Company, NY 1960, page 299.

Example:

```
LIB "phindex.lib";
ring r=0,(x(1..4)),ds;
poly Ax=4*x(1)^2+x(2)^2+x(3)^2+x(4)^2-4*x(1)*x(2)-4*x(1)*x(3)+4*x(1)*x(4)+4*x(2)*x(3)
signatureLqf(Ax,1); //The rank of Ax is 3+1=4
↦ 2,3,1
poly Bx=2*x(1)*x(4)+x(2)^2+x(3)^2;
signatureLqf(Bx);
↦ 2
```

D.14.6.3 PH_ais

Procedure from library `phindex.lib` (see Section D.14.6 [phindex_lib], page 2256).

Usage: PH_ais(I); I ideal of coordinates of the vector field.

Return: the Poincare-Hopf index of type int.

Note: the isolated singularity must be algebraically isolated.

Theory: The Poincare-Hopf index of a real vector field X at the isolated singularity 0 is the degree of the map (X/|X|) : S_epsilon —> S, where S is the unit sphere, and the spheres are oriented as (n-1)-spheres in R^n. The degree depends only on the germ, X, of X at 0. If the vector field X is real analytic, then an invariant of the germ is its local ring
Qx=R[[x1..xn]]/Ix
where R[[x1,..,xn]] is the ring of germs at 0 of real-valued analytic functions on R^n, and Ix is the ideal generated by the components of X. The isolated singularity of X is algebraically isolated if the algebra Qx is finite dimensional as real vector space, geometrically this mean that 0 is also an isolated singularity for the complexified vector field. In this case the Poincare-Hopf index is the signature of the non degenerate bilinear form <,> obtained by composition of the product in the algebra Qx with a linear functional map
<,> : (Qx)x(Qx) —(.)–> Qx —(L)–> R
with L(Jo)>0, where Jo is the residue class of the Jacobian determinant in Qx. Here, we use a natural linear functional defined as follows. Suppose that E={E_1,..E_r} is a basis of Qx, then Jo can be written as
Jo=a_1E_{j1}+...+a_kE_{jk}, js\in {1...r}, s=1..k, k<=r, where a_s are constant. The linear functional L:Qx—>R is defined as L(E_{j1})=(a_1)/|a_1|=sign of a_1,
the other elements of the base are sent to 0.
Refs. -Eisenbud & Levine, An algebraic formula for the degree of a C^\infty map germ, Ann. Math., 106, (1977), 19-38.
-Khimshiashvili, On a local degree of a smooth map, trudi Tbilisi Math. Inst., (1980), 105-124.

Example:

```
LIB "phindex.lib";
ring r=0,(x,y,z),ds;
ideal I=x3-3xy2,-y3+3yx2,z3;
PH_ais(I);
↦ 3
```

D.14.6.4 PH_nais

Procedure from library `phindex.lib` (see Section D.14.6 [phindex_lib], page 2256).

Usage: PH_nais(I); I ideal of coordinates of the vector field.

Return: the Poincare-Hopf index of type int.

Note: the vector field must be a non algebraically isolated singularity at 0, with reduced complex zeros of codimension 1.

Theory: Suppose that 0 is an algebraically isolated singularity of the real analytic vector field X, geometrically this corresponds to the fact that the complexified vector field has positive dimension singular locus, algebraically this mean that the local ring Qx=R[[x1..xn]]/Ix where R[[x1,..,xn]] is the ring of germs at 0 of real-valued analytic functions on R^n, and Ix is the ideal generated by the components of X is infinite dimensional as real vector space. In the case that X has a reduced hypersurface as complex zeros we have

the next. There exist a real analytic function f:R^n–>R, and a real analytic vector field Y s. t. X=fY. The function f does not change of sign out of 0 and
Mx=R[[x1..xn]]/(Ix : radical(Ix))
is a finite dimensional sub-algebra of Qx. The Poincare-Hopf index of X at 0 is the sign of f times the signature of the non degenerate bilinear form <,> obtained by composition of the product in the algebra Mx with a linear functional map
<,> : (Mx)x(Mx) —(.)–> Mx —(L)–> R
with L(Jp)>0, where Jp is the residue class of the Jacobian determinant of X, JX, over f^n, JX/(f^n) in Mx. Here, we use a natural linear functional defined as follows. Suppose that E={E_1,..E_r} is a basis of Mx, then Jp is writing as Jp=a_1E_{j1}+...+a_kE_{jk}, js\in {1...r}, s=1..k, k<=r, where a_s are constant. The linear functional L:M—>R is defined as L(E_{j1})=(a_1)/|a_1|=sign of a_1,
the other elements of the base are sent to 0.
Refs. -Castellanos-Vargas, V., Una formula algebraica del indice de Poincare-Hopf para campos vectoriales reales con una variedad de ceros complejos, Ph. D. thesis CIMAT (2000), chapther 1, Guanajuato Mexico.
-Castellanos -Vargas, V. The index of non algebraically isolated singularity, Bol. Soc. Mat. Mexicana, (3)
Vol. 8, 2002, 141-147.

Example:

```
LIB "phindex.lib";
ring r=0,(x,y,z),ds;
ideal I=x5-2x3y2-3xy4+x3z2-3xy2z2,-3x4y-2x2y3+y5-3x2yz2+y3z2,x2z3+y2z3+z5;
PH_nais(I);
↦ -3
```

D.14.7 polybori_lib

Library: polybori.lib

Purpose: A Singular Library Interface for PolyBoRi

Authors: Maximilian Kammermeier: Max0791@gmx.de
Susanne Scherer: sscherer90@yahoo.de

Overview: A library for using PolyBoRi in the SINGULAR interface, with procedures that convert structures (polynomials, rings, ideals) in both directions. Therefore, it is possible to compute boolean groebner basis via Section D.14.7.1 [boolean_std], page 2260. Polynomials can be converted to zero-supressed decision diagrams (zdd) and vice versa.

For usability it defines the PolyBoRi types bideal, bpoly, and bring which are equivalent to Singular's ideal, poly, and ring, as well as bset which corresponds to the type zdd introduced here. In addition bvar(i) constructs the Boolean variable corresponding to var(i) from current ring;

For convenience, the corresponding types can be converted explicitly or implicitly while assigning. Also several SINGULAR operators were overloaded: bring comes with nvars, bpoly implements lead, leadmonom and leadcoef. Objects of this type may be added and multiplied, too. Finally, bideal yields std and size as well as addition and element access.

Hence, by using these types PolyBoRi functionality can be carried out seamlessly in SINGULAR:

```
> LIB "polybori.lib";
> ring r0=2,x(1..4),lp;
> def x=bvar; // enforce Boolean variables

> bpoly f1=x(1)+x(4);
> bpoly f2=x(1)+x(3)*x(1);
> bideal bI=list(f1,f2);

> std(bI);
_[1] = x(1) + x(4)
_[2] = x(3)*x(4) + x(4)
```

Note: For using this library SINGULAR's `python` interface must be available on your system. Please `./configure --with-python` when building SINGULAR for this purpose.

There are prebuilt binary packages for `PolyBoRi` available from `http://polybori.sf.net/`.

For building your own `PolyBoRi` please ensure that you have `scons` and a development version of the boost libraries installed on you system. Then you may execute the following commands in a `bash`-style shell to build `PolyBoRi` available to `python`:

```
PBDIR=/path/to/custom/polybori
wget http://downloads.sf.net/project/polybori/polybori/\
0.8.2/polybori-0.8.2.tar.gz
tar -xvzf polybori-0.8.2.tar.gz
cd polybori-0.8.2
scons install PREFIX=$PBDIR PYINSTALLPREFIX=$PBDIR/python
export PYTHONPATH=$PBDIR/python:$PYTHONPATH
```

References:

See `http://polybori.sf.net` for details about `PolyBoRi`.

Procedures: See also: Section 3.8 [Libraries], page 55; Section 4.24 [User defined types], page 136; Section 4.28 [pyobject], page 140.

D.14.7.1 boolean_std

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_std(Is); Is ideal

Return: Singular ideal of the boolean groebner basis of Is

Example:

```
LIB "polybori.lib";
ring r0=2,x(1..4),lp;
poly f1=x(1)^2+2*x(2)*(x(3))-x(4)^3;
poly f2=x(1)^2-x(3)*x(1);
poly f3=x(2)+5-2*x(1);
poly f4=x(1)*x(2)-x(3);
ideal I=f1,f2,f3,f4;
boolean_std(I);         // implicitly add x(i)^2-x(i)
bideal bI=I;            // alternative syntax
bideal re = std(bI);  // Continue PolyBoRi computations
std(re[1..2]);
ring r1=0,x,Dp;
poly f1=x3+2*x+1;
poly f2=x10-x5+2x;
poly f3=19;
ideal I=f1,f2,f3;
boolean_std(I);
ring r2=32003,(x,y,z),Dp;
poly f1=xz+y+20*x^2*y;
poly f2=32002*xy+xz2+y;
ideal I=f1,f2;
boolean_std(I);
ring r2=32003,(x,y,z),Dp;
poly f1=xyz+20*x^2*y-3*xz+15;
poly f2=32002*xy+z2;
poly f3=19*x5y;
ideal I=f1,f2,f3;
boolean_std(I);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.4 [boolean_poly], page 2261; Section D.14.7.2 [boolean_poly_ring], page 2261; Section D.14.7.13 [from_boolean_ideal], page 2266; Section D.14.7.10 [from_boolean_poly], page 2264.

D.14.7.2 boolean_poly_ring

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

D.14.7.3 boolean_constant

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_constant(const[, rb]); const constant and rb boolean ring

Return: default: constant const in the representation of the boolean ring rb==boolean_poly_ring(basering); optional input: rb=boolean ring rb

Example:

```
LIB "polybori.lib";
ring r=7,(x,y),Dp;
pyobject rb=boolean_poly_ring(r);
boolean_constant(int(3));
typeof(boolean_constant(int(3)));
boolean_constant(int(0));
typeof(boolean_constant(int(0)));
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.4 boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_poly(ps[, dir, rb]); ps polynomial, dir integer zero or one, rb boolean ring

Return: default: polynomial ps in the representation of the boolean ring rb==boolean_poly_ring(basering); optional input: boolean ring rb

Note: via the optional input dir, one can choose the computation method (either direct[dir==0] or recursive[dir==1]). default: recursive

Example:

```
LIB "polybori.lib";
ring r=0,x(1..5),Dp;
poly f=x(2)*(x(3)-x(1))+x(4)*x(5);
bring rb=r;
boolean_poly(f);
boolean_poly(f,0);
boolean_poly(f,0,boolean_poly_ring(r));
boolean_poly(f,0,rb);
poly g=0;
boolean_poly(g);
poly g=1;
boolean_poly(g);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.5 direct_boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_poly(ps[, rb]); ps polynomial, rb boolean ring

Return: default: polynomial ps in the representation of the boolean ring rb==boolean_poly_ring(basering); optional input: boolean ring rb

Example:

```
LIB "polybori.lib";
ring r0=2,x(1..4),lp;
bring rb=r0;
poly f=x(1)^2+2*x(2)*(x(3))-x(4)^3;
direct_boolean_poly(f);
direct_boolean_poly(f,rb);
ring r1=0,x,Dp;
poly f=x3+2x+1;
direct_boolean_poly(f);
ring r2=32003,(x,y,z),Dp;
poly f=xyz+20*x^2*y-3*xz+15;
direct_boolean_poly(f);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.6 recursive_boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_poly(ps[, rb]); ps polynomial, rb boolean ring

Return: default: polynomial ps in the representation of the boolean ring rb==boolean_poly_ring(basering); optional input: rb boolean ring

Example:

```
LIB "polybori.lib";
ring r0=2,x(1..4),lp;
poly f=x(1)^2+2*x(2)*(x(3))-x(4)^3;
```

```
recursive_boolean_poly(f);
ring r1=0,x,Dp;
poly f=x3+2x+1;
recursive_boolean_poly(f);
ring r2=32003,(x,y,z),Dp;
def br2=boolean_poly_ring(r2);
bring bbr2=r2;
poly f=xyz+20*x^2*y-3*xz+15;
recursive_boolean_poly(f,br2);
recursive_boolean_poly(f,bbr2);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.7 boolean_ideal

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_ideal(Is[, rb]); Is Ideal, rb boolean ring

Return: default: ideal Is in the representation of the boolean ring rb==boolean_poly_ring(basering); optional input: rb boolean ring

Example:

```
LIB "polybori.lib";
ring r0=2,x(1..4),lp;
poly f1=x(1)^2+2*x(2)*(x(3))-x(4)^3;
poly f2=x(1)^2-x(3)*x(1);
poly f3=x(2)+5-2*x(1);
poly f4=x(1)*x(2)-x(3);
ideal I=f1,f2,f3,f4;
boolean_ideal(I);
ring r1=0,x,Dp;
poly f1=x3+2*x+1;
poly f2=x10-x5+2x;
poly f3=19;
ideal I=f1,f2,f3;
boolean_ideal(I);
ring r2=32003,(x,y,z),Dp;
bring bbr2=r2;
poly f1=xyz+20*x^2*y-3*xz+15;
poly f2=32002*xy+z2;
poly f3=19;
ideal I=f1,f2,f3;
boolean_ideal(I);
boolean_ideal(I,bbr2);
```

See also: Section D.14.7.1 [boolean_std], page 2260.

D.14.7.8 boolean_set

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: boolean_set(ss[, rb]); ss zdd, rb boolean ring

Return: default: boolean set ss in the representation of a Polybori boolean set in the ring rb==boolean_poly_ring(basering); optional input: boolean ring rb

Example:

```
LIB "polybori.lib";
ring rs=0,(x,y,z),Dp;
poly ps=(x+1)*(y+1)*(z+1);
zdd fz=ps;
boolean_set(fz);
poly g=x*y*z+1;
zdd gz=g;
boolean_set(gz);
ring R=0,(x(1..4)),Dp;
def Rb=boolean_poly_ring(R);
poly h=(x(1)+1)*(x(2)+1)*(x(3)+1)*(x(4)+1);
zdd hz=h;
boolean_set(hz);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.9 from_boolean_constant

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: from_boolean_constant(pb); pb pyobject

Return: constant polynomial

Example:

```
LIB "polybori.lib";
ring rs=0,(x,y,z),Dp;
def rsb=boolean_poly_ring(rs);
poly f=(x+y)*x+z;
bpoly pp=f;
from_boolean_constant(0);
from_boolean_constant(1);
from_boolean_constant(pp);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.10 from_boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: from_boolean_poly(ps[, dir]); ps polynomial, dir integer zero or one

Return: default: polynomial ps in the representation of the boolean ring

Note: via the optional input dir, one can choose the computation method (either direct[dir==0] or recursive[dir==1]). default: direct

Example:

```
LIB "polybori.lib";
ring r=0,(x,y,z),Dp;
def rb=boolean_poly_ring(r);
poly f=x^2+2*y+5*z^4;
bpoly pp=f;
from_boolean_poly(pp);
from_boolean_poly(pp,1);
ring r2=5,(x,y,z),Dp;
```

```
def rb2=boolean_poly_ring(r2);
poly f2=x+y+z;
bpoly pp2=f2;
from_boolean_poly(pp);
from_boolean_poly(pp,1);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.11 direct_from_boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: from_boolean_poly(pb); pb boolean polynomial

Return: polynomial in Singular

Example:

```
LIB "polybori.lib";
ring r=0,(x,y,z),Dp;
pyobject rb=boolean_poly_ring(r);
poly f=x^2+2*y^3+3*z^3;
bpoly pp=f;
direct_from_boolean_poly(pp);
poly g=0;
bpoly pp=g;
direct_from_boolean_poly(pp);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.4 [boolean_poly], page 2261; Section D.14.7.2 [boolean_poly_ring], page 2261; Section D.14.7.1 [boolean_std], page 2260; Section D.14.7.13 [from_boolean_ideal], page 2266.

D.14.7.12 recursive_from_boolean_poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: recursive_from_boolean_poly(pb); pb boolean polynomial

Return: polynomial in Singular

Example:

```
LIB "polybori.lib";
ring rs=0,(x,y,z),Dp;
def rsb=boolean_poly_ring(rs);
poly f=(x+y)*x+z;
bpoly pp=f;
recursive_from_boolean_poly(pp);
ring rs2=2,(x,y,z),Dp;
def rsb2=boolean_poly_ring(rs2);
poly f2=(x+y)*x+x;
bpoly pp2=f2;
recursive_from_boolean_poly(pp);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.4 [boolean_poly], page 2261; Section D.14.7.2 [boolean_poly_ring], page 2261; Section D.14.7.13 [from_boolean_ideal], page 2266; Section D.14.7.10 [from_boolean_poly], page 2264.

D.14.7.13 from_boolean_ideal

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: from_boolean_ideal(I); I boolean ideal

Return: ideal in Singular

Example:

```
LIB "polybori.lib";
ring rs=0,(x,y,z),Dp;
def rb3=boolean_poly_ring(rs);
poly f1=x+y;
poly f2=x+z;
bpoly pp =f1;
bpoly p = f2;
bideal Ib;
list K=(p,pp);
Ib=K;
from_boolean_ideal(Ib);
ring rs2=5,(x,y,z),Dp;
def rb4=boolean_poly_ring(rs2);
poly p1=x+y;
poly p2=x+z;
poly p3=y+z;
bpoly p = p1;
bpoly pp = p2;
bpoly ppp = p3;
bideal Ib;
list K=(p,pp,ppp);
Ib=K;
from_boolean_ideal(Ib);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.4 [boolean_poly], page 2261; Section D.14.7.2 [boolean_poly_ring], page 2261; Section D.14.7.1 [boolean_std], page 2260; Section D.14.7.10 [from_boolean_poly], page 2264.

D.14.7.14 from_boolean_set

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: from_boolean_set(sb); sb boolean set

Return: Boolean set sb in the representation of a zdd

Example:

```
LIB "polybori.lib";
ring r=0,(x,y,z),Dp;
poly f=(x+1)*(y+1)*(z+1);
bpoly fb=f;
bset fs=fb;
from_boolean_set(fs);
poly g=x*y*z+1;
bpoly gb=g;
bset gs=gb;
from_boolean_set(gs);
```

```
     ring R=0,(x(1..4)),Dp;
     poly h=(x(1)+1)*(x(2)+1)*(x(3)+1)*(x(4)+1);
     pyobject hb=boolean_poly(h);
     def hs=hb.set();
     from_boolean_set(hs);
```

See also: Section D.14.7.7 [boolean_ideal], page 2263; Section D.14.7.1 [boolean_std], page 2260.

D.14.7.15 bvar

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: bvar(i); int i

Return: i-th variable of Boolean ring corresponding to current basering

Example:

```
     LIB "polybori.lib";
     ring r = 2,(x,y,z),Dp;
     bvar(1); // -> x
```

See also: Section D.14.7.2 [boolean_poly_ring], page 2261; Section 5.1.165 [var], page 285.

D.14.7.16 poly2zdd

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: poly2zdd(poly ps); polynomial ps

Return: polynomial ps in zdd representation

Example:

```
     LIB "polybori.lib";
     ring r=0,x(1..5),Dp;
     poly f=(x(1)+1)*(x(2)+1)*(x(3)+1)*x(4)*x(5);
     poly2zdd(f);
     poly g=x(3);
     poly2zdd(g);
```

See also: Section D.14.7.4 [boolean_poly], page 2261; Section D.14.7.14 [from_boolean_set], page 2266.

D.14.7.17 zdd2poly

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: zdd2poly(ss); zero-supressed decision diagram ss

Return: zdd ss in polynomial representation

Example:

```
     LIB "polybori.lib";
     ring r=0,x(1..5),Dp;
     poly f=(x(1)+1)*(x(2)+1)*(x(3)+1)*x(4)*x(5);
     zdd2poly(poly2zdd(f));
     poly g=x(3);
     zdd2poly(poly2zdd(g));
     poly g=0;
     zdd2poly(poly2zdd(0));
```

```
poly g=1;
zdd2poly(poly2zdd(01));
```

See also: Section D.14.7.8 [boolean_set], page 2263; Section D.14.7.10 [from_boolean_poly], page 2264.

D.14.7.18 disp_zdd

Procedure from library `polybori.lib` (see Section D.14.7 [polybori_lib], page 2259).

Usage: disp_zdd(ss); zero-supressed decision diagram ss

Return: string containing visualization of ss

Note: the resulting string is the visualization of the polynomial that corresponds to ss, but with a additional structure that comes from the zdd. Every reached else- Branch induces a new line in the string.

Example:

```
LIB "polybori.lib";
ring r1=0,(x,y,z),Dp;
poly f1=xyz+xy+xz+yz+y+z+x+1;
zdd s1=f1;
disp_zdd(s1);
ring r2=0,x(1..6),Dp;
poly f2=x(1)+x(2)+x(3)+x(5)^2+x(6);
zdd s2=f2;
disp_zdd(s2);
ring r4=0,x(1..6),Dp;
poly f2=x(1)+1;
zdd s2=f2;
disp_zdd(s2);
ring r2=0,x(1..6),Dp;
poly f2=x(1)*x(2)*(x(3)-x(5)^2*x(6))+3*x(4)*x(5)-3;
zdd s2=f2;
disp_zdd(s2);
poly f4=0;
zdd s4=f4;
disp_zdd(s4);
poly f5=1;
zdd s5=f5;
disp_zdd(s5);
```

See also: Section D.14.7.16 [poly2zdd], page 2267; Section D.14.7.17 [zdd2poly], page 2267.

D.14.8 sets_lib

Library: sets.lib

Purpose: Sets in Singular

Authors: J. Boehm, boehm @ mathematik.uni-kl.de
D. Wienholz, wienholz @ mathematik.uni-kl.de
S. Zillien, zillien @ rhrk.uni-kl.de

Overview: We implement the new class set and all basic methods needed to work with sets. A set is generated from a list. After the generating of a set, the adding of an element or the union of two sets, automatically every double element is removed to secure that no element occurs in a set more than once.

There is a comparison operator, we access the operator via the function isEqual. This function isEqual can be used to compare two elements of the same type (Set, list, int, bigint, string, intmat, bigintmat, intvec, ring, map, poly, matrix, ideal, module, vector, resolution) and also works for comparing of int, bigint and number with each other, similarly for matrix, bigintmat and intmat.

The function size can be used to determine the number of elements.

The + operator is used for the union, the * operator for the intersection.

The operators < and > can be used for inclusion tests.

The print function can be used for printing sets.

Note that the implementation of the underlying data structure and algorithms is very trivial and will at some point be replaced with something more efficient.

Types: Set The class of all sets

Procedures:

D.14.8.1 set

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: set(l) or *=l (short form of * = set(l)); l list

Return: Set, the set createt from the entered list

Example:

```
LIB "sets.lib";
//example for set
Set S0a = list(list(1,2,3),list(list(1,2)),list(10,11));
Set S0b = list(list(10,11),list(list(1,2)));
S0b<S0a;
↦ 1
S0a<S0b;
↦ 0
S0a==S0a;
↦ 1
S0a==S0b;
↦ 0
list L = 1,1,2,3;
Set S1 = L;
S1;
↦ {3; 2; 1}
↦ Set with 3 elements
↦
ring R1;
ring R2 = 0,(x,y),dp;
Set S2 = list(R1,R1,R2);
S2;
↦ {(QQ),(x,y),(dp(2),C); (ZZ/32003),(x,y,z),(dp(3),C)}
↦ Set with 2 elements
↦
ideal I1 = x+y;
ideal I2 = y^2;
ideal I3 = x+y, (x+y)^3;
Set S3 = list(I1,I2,I3);
S3;
↦ {y2; x+y}
↦ Set with 2 elements
↦
isEqual(I1,I3);
↦ 1
isEqual(I1,I2);
↦ 0
module M1 = x*gen(1), y*gen(1);
module M2 = y^2*gen(2);
module M3 = (x+y)*gen(1), (x-y)*gen(1);
Set S4 = list(M1,M2,M3);
S4;
↦ {y2*gen(2); x*gen(1),y*gen(1)}
↦ Set with 2 elements
↦
intmat m1[2][3]= 1,2,3,4,5,6;
intmat m2[2][3]= 1,2,3,4,5,7;
Set S5 = list(m1,m2,m1);
S5;
↦ {1,2,3,4,5,7 ; 1,2,3,4,5,6 }
↦ Set with 2 elements
↦
bigintmat b1[2][3]= 1,2,3,4,5,6;
bigintmat b2[2][3]= 1,2,3,4,5,7;
Set S6 = list(b1,b2,b1);
S6;
↦ {1, 2, 3, 4, 5, 7; 1, 2, 3, 4, 5, 6}
↦ Set with 2 elements
↦
resolution r1 = res(maxideal(3),0);
resolution r2 = res(maxideal(4),0);
Set S7 = list(r1,r1,r2);
size(S7);
↦ 2
```

D.14.8.2 union

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: union(N,M) or N+M; N,M sets

Return: Set, the union of the sets N and M

Example:

```
LIB "sets.lib";
list l =1,2,3;
list j =2,3,4;
Set N=l;
Set M=j;
N;
```

```
↦ {3; 2; 1}
↦ Set with 3 elements
↦
M;
↦ {4; 3; 2}
↦ Set with 3 elements
↦
N+M;
↦ {1; 4; 3; 2}
↦ Set with 4 elements
↦
```

D.14.8.3 intersectionSet

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: intersectionSet(N,M) or N*M; N,M sets

Return: Set, the interseection of the sets N and M

Example:

```
LIB "sets.lib";
list l =1,2,3;
list j =2,3,4;
Set N=l;
Set M=j;
N;
↦ {3; 2; 1}
↦ Set with 3 elements
↦
M;
↦ {4; 3; 2}
↦ Set with 3 elements
↦
N*M;
↦ {3; 2}
↦ Set with 2 elements
↦
```

D.14.8.4 complement

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: complement(N,M); N,M sets

Return: Set, the complement of the set N in M

Example:

```
LIB "sets.lib";
list l =1,2;
list j =1,2,3,4;
Set N=l;
Set M=j;
N;
↦ {2; 1}
↦ Set with 2 elements
```

```
↦
M;
↦ {4; 3; 2; 1}
↦ Set with 4 elements
↦
complement(N,M);
↦ {3; 4}
↦ Set with 2 elements
↦
```

D.14.8.5 isElement

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: isElement(a,M); M set a def

Return: bool, 1 if a is an element of M, 0 if not

Example:

```
LIB "sets.lib";
int i=1;
int j=5;
list k =1,2,3,4;
Set M=k;
i;
↦ 1
j;
↦ 5
M;
↦ {4; 3; 2; 1}
↦ Set with 4 elements
↦
isElement(i,M);
↦ 1
isElement(j,M);
↦ 0
```

D.14.8.6 isSubset

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: isSubset(N,M) or N<M ; N,M sets

Return: bool, 1 if N is a Subset of M or 0 if not

Example:

```
LIB "sets.lib";
list l =1,2;
list j =1,2,3,4;
Set N=l;
Set M=j;
N;
↦ {2; 1}
↦ Set with 2 elements
↦
M;
```

```
↦ {4; 3; 2; 1}
↦ Set with 4 elements
↦
N<M;
↦ 1
M<N;
↦ 0
```

D.14.8.7 isSuperset

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Return: bool, 1 if N is a Superset of M or 0 if not

Example:

```
LIB "sets.lib";
list l =1,2;
list j =1,2,3,4;
Set N=l;
Set M=j;
N;
↦ {2; 1}
↦ Set with 2 elements
↦
M;
↦ {4; 3; 2; 1}
↦ Set with 4 elements
↦
N>M;
↦ 0
M>N;
↦ 1
```

D.14.8.8 addElement

Procedure from library `sets.lib` (see Section D.14.8 [sets_lib], page 2268).

Usage: addElement(M,a) ; M Set, a freely chosen element

Return: adds the Element a to the Set M

Example:

```
LIB "sets.lib";
//example for addElement
int a=4;
list L = 1,2,3;
Set S = L;
S;
↦ {3; 2; 1}
↦ Set with 3 elements
↦
a;
↦ 4
addElement(S,a);
↦ {1; 2; 3; 4}
↦ Set with 4 elements
↦
```

D.15 Experimental libraries

This sections collect libraries in the beta test phase. Everything in these libraries may change.

For the minimal requirements and guidelines see Section 3.8 [Libraries], page 55.

Comments should be send to the author of the library directly.

D.15.1 autgradalg_lib

Library: autgradalg.lib

Purpose: Compute automorphism groups of pointedly graded algebras and of Mori dream spaces.

Authors: Simon Keicher

Overview: This library provides a framework for computing automorphisms of integral, finitely generated algebras that are graded by a finitely generated abelian group. This library also contains functions to compute automorphism groups of Mori dream spaces. The results are ideals I such that the respective automorphism group is isomorphic to the subgroup V(I) in some GL(n).

Assumptions:
* the algebra R is given as factor algebra S/I with a graded polynomial ring S = KK[T_1,...,T_r]. We will always assume that the basering is S and it is given over the rationals QQ or a number field QQ(a). * R must be minimally presented, i.e., I is contained in <T_1,...,T_r>^2. * S (and hence R) are graded via ’setBaseMultigrading(Q)’ from ’multigrading.lib’. The last rows of the matrix Q are interpreted over ZZ/a_iZZ if the respective entry of the list TOR is a_i and has been provided as parameter to the respective function. (See the functions for more details.) * For all 1 <= i <= r: I_{w_i} = 0 where w_i := deg(T_i). * the grading is pointed, i.e., no generator has degree 0 and the cone over all generator degrees is pointed. * For Mori dream spaces X, we assume them to be given as X = X(R,w) with the Cox ring R of X (given as the algebra R before) and an ample class w in the grading group K with the torsion entries removed.

Note: we require the user to execute ’LIB’gfanlib.so” before using this library.

Procedures:

Note: the following functions were taken from ’quotsingcox.lib’ by M.Donten-Bury and S.Keicher: ’hilbertBas’.

Note: This library comes without any warranty whatsoever. Use it at your own risk.

D.15.1.1 autKS

Procedure from library `autgradalg.lib` (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: autKS(TOR); TOR: optional list of elementary divisors in case of torsion.

Assume: the basering is multigraded having used the command setBaseMultigrading(Q) from ’multigrading.lib’.

Purpose: Compute the subgroup Aut_K(S) of GL(n) of graded automorphisms of the polynomial ring S (the basering).

Return: returns a ring S and exports an ideal ideal Iexported in the coordinate ring S = K[Y_ij] of GL(n) such that Aut_K(S) = V(I).

Example:

```
LIB "autgradalg.lib";
//////////////////
// example: fano 15:
intmat Q[1][5] = 3,3,2,2,1;
ring R = 0,T(1..5),dp;
// attach degree matrix Q to R:
setBaseMultigrading(Q);
//ideal I = T(1)*T(2) + T(3)^2*T(4) + T(5)^6;
def S = autKS();
↦ // coefficients: QQ
↦ // number of vars : 50
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29\
   ) Y(30) Y(31) Y(32) Y(33) Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41)\
    Y(42) Y(43) Y(44) Y(45)
↦ //        block   2 : ordering C
setring S;
dim(std(Iexported));
↦ 17
basering;
↦ // coefficients: QQ
↦ // number of vars : 46
↦ //        block   1 : ordering dp
↦ //                  : names    Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(\
   9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21\
   ) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29) Y(30) Y(31) Y(32) Y(33)\
    Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41) Y(42) Y(43) Y(44) Y(45)
↦ //        block   2 : ordering dp
↦ //                  : names    Z
↦ //        block   3 : ordering C
autKSexported;
↦ [1]:
↦    [1]:
↦       _[1,1]=Y(1)
↦       _[1,2]=Y(2)
↦       _[1,3]=0
↦       _[1,4]=0
↦       _[1,5]=0
↦       _[1,6]=Y(6)
↦       _[1,7]=Y(7)
↦       _[1,8]=0
↦       _[1,9]=Y(9)
↦       _[2,1]=Y(10)
↦       _[2,2]=Y(11)
↦       _[2,3]=0
↦       _[2,4]=0
↦       _[2,5]=0
↦          _[2,6]=Y(15)
↦          _[2,7]=Y(16)
↦          _[2,8]=0
↦          _[2,9]=Y(18)
↦          _[3,1]=0
↦          _[3,2]=0
↦          _[3,3]=Y(21)
↦          _[3,4]=Y(22)
↦          _[3,5]=0
↦          _[3,6]=0
↦          _[3,7]=0
↦          _[3,8]=Y(26)
↦          _[3,9]=0
↦          _[4,1]=0
↦          _[4,2]=0
↦          _[4,3]=Y(30)
↦          _[4,4]=Y(31)
↦          _[4,5]=0
↦          _[4,6]=0
↦          _[4,7]=0
↦          _[4,8]=Y(35)
↦          _[4,9]=0
↦          _[5,1]=0
↦          _[5,2]=0
↦          _[5,3]=0
↦          _[5,4]=0
↦          _[5,5]=Y(41)
↦          _[5,6]=0
↦          _[5,7]=0
↦          _[5,8]=0
↦          _[5,9]=0
↦          _[6,1]=0
↦          _[6,2]=0
↦          _[6,3]=0
↦          _[6,4]=0
↦          _[6,5]=0
↦          _[6,6]=Y(21)*Y(41)
↦          _[6,7]=Y(22)*Y(41)
↦          _[6,8]=0
↦          _[6,9]=Y(26)*Y(41)
↦          _[7,1]=0
↦          _[7,2]=0
↦          _[7,3]=0
↦          _[7,4]=0
↦          _[7,5]=0
↦          _[7,6]=Y(30)*Y(41)
↦          _[7,7]=Y(31)*Y(41)
↦          _[7,8]=0
↦          _[7,9]=Y(35)*Y(41)
↦          _[8,1]=0
↦          _[8,2]=0
↦          _[8,3]=0
↦          _[8,4]=0
↦         _[8,5]=0
↦         _[8,6]=0
↦         _[8,7]=0
↦         _[8,8]=Y(41)^2
↦         _[8,9]=0
↦         _[9,1]=0
↦         _[9,2]=0
↦         _[9,3]=0
↦         _[9,4]=0
↦         _[9,5]=0
↦         _[9,6]=0
↦         _[9,7]=0
↦         _[9,8]=0
↦         _[9,9]=Y(41)^3
↦      [2]:
↦         1
↦      [3]:
↦         _[1]=Y(3)
↦         _[2]=Y(4)
↦         _[3]=Y(5)
↦         _[4]=Y(8)
↦         _[5]=Y(12)
↦         _[6]=Y(13)
↦         _[7]=Y(14)
↦         _[8]=Y(17)
↦         _[9]=Y(19)
↦         _[10]=Y(20)
↦         _[11]=Y(23)
↦         _[12]=Y(24)
↦         _[13]=Y(25)
↦         _[14]=Y(27)
↦         _[15]=Y(28)
↦         _[16]=Y(29)
↦         _[17]=Y(32)
↦         _[18]=Y(33)
↦         _[19]=Y(34)
↦         _[20]=Y(36)
↦         _[21]=Y(37)
↦         _[22]=Y(38)
↦         _[23]=Y(39)
↦         _[24]=Y(40)
↦         _[25]=Y(42)
↦         _[26]=Y(43)
↦         _[27]=Y(44)
↦         _[28]=Y(45)
↦         _[29]=-Y(2)*Y(10)*Y(22)^2*Y(30)^2*Y(41)^8*Z+Y(1)*Y(11)*Y(22)^2*Y(30\
   )^2*Y(41)^8*Z+2*Y(2)*Y(10)*Y(21)*Y(22)*Y(30)*Y(31)*Y(41)^8*Z-2*Y(1)*Y(11)\
   *Y(21)*Y(22)*Y(30)*Y(31)*Y(41)^8*Z-Y(2)*Y(10)*Y(21)^2*Y(31)^2*Y(41)^8*Z+Y\
   (1)*Y(11)*Y(21)^2*Y(31)^2*Y(41)^8*Z-1
↦      [4]:
↦         T(1),T(2),T(3),T(4),T(5),T(3)*T(5),T(4)*T(5),T(5)^2,T(5)^3
getVariableWeights();
↦ 3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3\
   ,3,2,2,1,3,3,2,3,-22
kill S, Q, R;
/////////////
// example 3.14 from the paper
intmat Q[3][5] =
1,1,1,1,1,
1,-1,0,0,1,
1,1,1,0,0;
list TOR = 2;
ring R = 0,T(1..5),dp;
// attach degree matrix Q to R:
setBaseMultigrading(Q);
//ideal I = T(1)*T(2) + T(3)^2 + T(4)^2;
def S = autKS();
↦ // coefficients: QQ
↦ // number of vars : 30
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25)
↦ //        block   2 : ordering C
↦ // ** redefining adMons (  list adMons;) autgradalg.lib::autKS:2325
↦ // ** redefining MONexported (  export(MONexported);)
setring S;
Iexported;
↦ Iexported[1]=Y(23)
↦ Iexported[2]=Y(22)
↦ Iexported[3]=Y(21)
↦ Iexported[4]=Y(18)
↦ Iexported[5]=Y(17)
↦ Iexported[6]=Y(16)
↦ Iexported[7]=Y(15)
↦ Iexported[8]=Y(14)
↦ Iexported[9]=Y(12)
↦ Iexported[10]=Y(11)
↦ Iexported[11]=Y(10)
↦ Iexported[12]=Y(9)
↦ Iexported[13]=Y(8)
↦ Iexported[14]=Y(5)
↦ Iexported[15]=Y(4)
↦ Iexported[16]=Y(3)
↦ Iexported[17]=Y(24)*Y(25)
↦ Iexported[18]=Y(20)*Y(25)
↦ Iexported[19]=Y(6)*Y(25)
↦ Iexported[20]=Y(2)*Y(25)
↦ Iexported[21]=Y(19)*Y(24)
↦ Iexported[22]=Y(7)*Y(24)
↦ Iexported[23]=Y(1)*Y(24)
↦ Iexported[24]=Y(19)*Y(20)
↦ Iexported[25]=Y(7)*Y(20)
↦ Iexported[26]=Y(1)*Y(20)
↦ Iexported[27]=Y(6)*Y(19)
↦ Iexported[28]=Y(2)*Y(19)
↦ Iexported[29]=Y(6)*Y(7)
↦ Iexported[30]=Y(2)*Y(7)
↦ Iexported[31]=Y(1)*Y(6)
↦ Iexported[32]=Y(1)*Y(2)
↦ Iexported[33]=Y(2)*Y(6)*Y(13)*Y(20)*Y(24)*Z+Y(1)*Y(7)*Y(13)*Y(19)*Y(25)*Z\
   -1
↦ Iexported[34]=Y(1)*Y(7)*Y(13)*Y(19)*Y(25)^2*Z-Y(25)
↦ Iexported[35]=Y(1)*Y(7)*Y(13)*Y(19)^2*Y(25)*Z-Y(19)
↦ Iexported[36]=Y(1)*Y(7)^2*Y(13)*Y(19)*Y(25)*Z-Y(7)
↦ Iexported[37]=Y(1)^2*Y(7)*Y(13)*Y(19)*Y(25)*Z-Y(1)
print(getVariableWeights());
↦      1     1     1     1     1     1     1     1     1     1     1     1 \
       1     1     1     1     1     1     1     1     1     1     1     1  \
      1    -5
↦      1    -1     0     0     1     1    -1     0     0     1     1    -1 \
       0     0     1     1    -1     0     0     1     1    -1     0     0  \
      1    -1
↦      1     1     1     0     0     1     1     1     0     0     1     1 \
       1     0     0     1     1     1     0     0     1     1     1     0  \
      0    -3
kill S, R, Q;
```

D.15.1.2 autGradAlg

Procedure from library `autgradalg.lib` (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: autGradAlg(I, TOR); I is an ideal, TOR is an optional list of integers representing the torsion part of the grading group.

Assume: minimally presented, degrees of the generators of I
are the minimal degrees, basering multigraded pointedly, I_w = 0 for all w = deg(var(i))

Return: a ring. Also exports an ideal Jexported and a list stabExported.

Example:

```
LIB "autgradalg.lib";
intmat Q[1][3] =
1,1,1;
ring R = 0,T(1..3), dp;
setBaseMultigrading(Q);
ideal I = 0; //T(1)*T(2) + T(3)*T(4);
def RR = autGradAlg(I);
↦ // coefficients: QQ
↦ // number of vars : 12
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) Y(1) Y(2) Y(3) Y(4) Y(5) Y(\
   6) Y(7) Y(8) Y(9)
↦ //        block   2 : ordering C
↦ // ** redefining adMons (  list adMons;) autgradalg.lib::autKS:2324
setring RR;
"resulting ideal:";
↦ resulting ideal:
Jexported;
↦ Jexported[1]=-Y(3)*Y(5)*Y(7)*Z+Y(2)*Y(6)*Y(7)*Z+Y(3)*Y(4)*Y(8)*Z-Y(1)*Y(6\
   )*Y(8)*Z-Y(2)*Y(4)*Y(9)*Z+Y(1)*Y(5)*Y(9)*Z-1
```

```
"dimension:";
↦ dimension:
dim(std(Jexported));
↦ 9
"as a detailed list:";
↦ as a detailed list:
stabExported;
↦ [1]:
↦    [1]:
↦       _[1,1]=Y(1)
↦       _[1,2]=Y(2)
↦       _[1,3]=Y(3)
↦       _[2,1]=Y(4)
↦       _[2,2]=Y(5)
↦       _[2,3]=Y(6)
↦       _[3,1]=Y(7)
↦       _[3,2]=Y(8)
↦       _[3,3]=Y(9)
↦    [2]:
↦       1
↦    [3]:
↦       _[1]=-Y(3)*Y(5)*Y(7)*Z+Y(2)*Y(6)*Y(7)*Z+Y(3)*Y(4)*Y(8)*Z-Y(1)*Y(6)*\
   Y(8)*Z-Y(2)*Y(4)*Y(9)*Z+Y(1)*Y(5)*Y(9)*Z-1
```

D.15.1.3 autGenWeights

Procedure from library `autgradalg.lib` (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: autGenWeights(Q): Q is an intmat (columns must contain a lattice basis).

Assume: the cone over Q must be pointed and the columns of Q contain a lattice basis; there must be no 0-columns in Q. We assume that, in the torsion case, the torsion rows of Q are reduced (for example, a row of Q standing for entries in ZZ/5ZZ must not contain elements > 5 or < 0).

Purpose: computes generators for the subgroup aut(Omega_S) of GL(n, \ZZ) that consists of all invertible integer kxk matrices which fix the set Omega_S of degrees of the variables of the basering S. The set of columns of Q equals Omega_S.

Reference:

Remark 3.1.

Return: a list of integral matrices A with |det A| = 1 such that A*{columns of Q} = {columns of Q}.

Example:

```
LIB "autgradalg.lib";
// torsion example
// ZZ + ZZ/5ZZ:
intmat Q[2][5] =
1,1,1,1,1,
2,3,1,4,0;
list TOR = 5;
autGenWeights(Q, TOR);
↦ [1]:
```

```
↦     1,0,
↦     0,1
↦  [2]:
↦     1,0,
↦     0,2
↦  [3]:
↦     1,0,
↦     0,3
↦  [4]:
↦     1,0,
↦     0,4
↦  [5]:
↦     1,0,
↦     1,1
↦  [6]:
↦     1,0,
↦     1,2
↦  [7]:
↦     1,0,
↦     1,3
↦  [8]:
↦     1,0,
↦     1,4
↦  [9]:
↦     1,0,
↦     2,1
↦  [10]:
↦     1,0,
↦     2,2
↦  [11]:
↦     1,0,
↦     2,3
↦  [12]:
↦     1,0,
↦     2,4
↦  [13]:
↦     1,0,
↦     3,1
↦  [14]:
↦     1,0,
↦     3,2
↦  [15]:
↦     1,0,
↦     3,3
↦  [16]:
↦     1,0,
↦     3,4
↦  [17]:
↦     1,0,
↦     4,1
↦  [18]:
↦     1,0,
↦     4,2
↦ [19]:
↦    1,0,
↦    4,3
↦ [20]:
↦    1,0,
↦    4,4
kill Q, TOR;
// another free example
intmat Q[2][6] =
-2,2,-1,1,-1,1,
1,1,1,1,1,1;
autGenWeights(Q);
↦ [1]:
↦    1,0,
↦    0,1
↦ [2]:
↦    -1,0,
↦    0,1
kill Q;
//-----------------
// 2nd free example
intmat Q[2][4] =
1,0,1,1,
0,1,1,1;
autGenWeights(Q);
↦ [1]:
↦    1,0,
↦    0,1
↦ [2]:
↦    0,1,
↦    1,0
kill Q;
```

D.15.1.4 stabilizer

Procedure from library **autgradalg.lib** (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: stabilizer(RL, A, BB, AMON, n0): RL is an ideal, A a matrix (standing for a subgroup of GL(n)), BB is an intmat (standing for an automorphism of the grading group), AMON a list of monomials corresponding to the rows/columns of A, n0 an integer such that the first n0 variables of the basering are the T(i), optional: a list of elementary divisors if there is torsion.

Assume: the basering must be graded (see setBaseMultigrading()) and the cone over the degrees of the variables must be pointed; there mustn't be 0-degrees. The vector w must be an element of the cone over the degrees of the variables. Moreover, B must be such that it permutes the degrees of the variables and the degrees of the generators of RL.

Purpose: returns relations such that A*I = I.

Return: a ring. Also exports an ideal Jexported and a list stabExported.

Example:

```
LIB "autgradalg.lib";
//////////////////
```

```
// example: fano 15:
intmat Q[1][5] = 3,3,2,2,1;
ring R = 0,T(1..5),dp;
// attach degree matrix Q to R:
setBaseMultigrading(Q);
ideal I = T(1)*T(2) + T(3)^2*T(4) + T(5)^6;
def RR = stabilizer(I);
↦ // coefficients: QQ
↦ // number of vars : 50
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29\
   ) Y(30) Y(31) Y(32) Y(33) Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41)\
    Y(42) Y(43) Y(44) Y(45)
↦ //        block   2 : ordering C
setring RR;
RR;
↦ // coefficients: QQ
↦ // number of vars : 46
↦ //        block   1 : ordering dp
↦ //                  : names    Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(\
   9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21\
   ) Y(22) Y(23) Y(24) Y(25) Y(26) Y(27) Y(28) Y(29) Y(30) Y(31) Y(32) Y(33)\
    Y(34) Y(35) Y(36) Y(37) Y(38) Y(39) Y(40) Y(41) Y(42) Y(43) Y(44) Y(45)
↦ //        block   2 : ordering dp
↦ //                  : names    Z
↦ //        block   3 : ordering C
Jexported;
↦ Jexported[1]=Y(3)
↦ Jexported[2]=Y(4)
↦ Jexported[3]=Y(5)
↦ Jexported[4]=Y(8)
↦ Jexported[5]=Y(12)
↦ Jexported[6]=Y(13)
↦ Jexported[7]=Y(14)
↦ Jexported[8]=Y(17)
↦ Jexported[9]=Y(19)
↦ Jexported[10]=Y(20)
↦ Jexported[11]=Y(23)
↦ Jexported[12]=Y(24)
↦ Jexported[13]=Y(25)
↦ Jexported[14]=Y(27)
↦ Jexported[15]=Y(28)
↦ Jexported[16]=Y(29)
↦ Jexported[17]=Y(32)
↦ Jexported[18]=Y(33)
↦ Jexported[19]=Y(34)
↦ Jexported[20]=Y(36)
↦ Jexported[21]=Y(37)
↦ Jexported[22]=Y(38)
↦ Jexported[23]=Y(39)
↦ Jexported[24]=Y(40)
↦ Jexported[25]=Y(42)
↦ Jexported[26]=Y(43)
↦ Jexported[27]=Y(44)
↦ Jexported[28]=Y(45)
↦ Jexported[29]=-Y(2)*Y(10)*Y(22)^2*Y(30)^2*Y(41)^8*Z+Y(1)*Y(11)*Y(22)^2*Y(\
   30)^2*Y(41)^8*Z+2*Y(2)*Y(10)*Y(21)*Y(22)*Y(30)*Y(31)*Y(41)^8*Z-2*Y(1)*Y(1\
   1)*Y(21)*Y(22)*Y(30)*Y(31)*Y(41)^8*Z-Y(2)*Y(10)*Y(21)^2*Y(31)^2*Y(41)^8*Z\
   +Y(1)*Y(11)*Y(21)^2*Y(31)^2*Y(41)^8*Z-1
↦ Jexported[30]=Y(26)^2*Y(31)+2*Y(22)*Y(26)*Y(35)+Y(9)*Y(16)+Y(7)*Y(18)
↦ Jexported[31]=2*Y(22)*Y(26)*Y(31)+Y(22)^2*Y(35)+Y(7)*Y(16)
↦ Jexported[32]=Y(22)^2*Y(31)
↦ Jexported[33]=Y(26)^2*Y(30)+2*Y(21)*Y(26)*Y(35)+Y(9)*Y(15)+Y(6)*Y(18)
↦ Jexported[34]=2*Y(22)*Y(26)*Y(30)+2*Y(21)*Y(26)*Y(31)+2*Y(21)*Y(22)*Y(35)\
   +Y(7)*Y(15)+Y(6)*Y(16)
↦ Jexported[35]=Y(22)^2*Y(30)+2*Y(21)*Y(22)*Y(31)
↦ Jexported[36]=2*Y(21)*Y(26)*Y(30)+Y(21)^2*Y(35)+Y(6)*Y(15)
↦ Jexported[37]=-Y(41)^6+2*Y(21)*Y(22)*Y(30)+Y(21)^2*Y(31)-Y(26)^2*Y(35)-Y(\
   9)*Y(18)
↦ Jexported[38]=Y(21)^2*Y(30)
↦ Jexported[39]=Y(9)*Y(11)+Y(2)*Y(18)
↦ Jexported[40]=Y(7)*Y(11)+Y(2)*Y(16)
↦ Jexported[41]=Y(6)*Y(11)+Y(2)*Y(15)
↦ Jexported[42]=Y(2)*Y(11)
↦ Jexported[43]=Y(9)*Y(10)+Y(1)*Y(18)
↦ Jexported[44]=Y(7)*Y(10)+Y(1)*Y(16)
↦ Jexported[45]=Y(6)*Y(10)+Y(1)*Y(15)
↦ Jexported[46]=-2*Y(21)*Y(22)*Y(30)-Y(21)^2*Y(31)+Y(2)*Y(10)+Y(1)*Y(11)
↦ Jexported[47]=Y(1)*Y(10)
dim(std(Jexported));
↦ 3
getVariableWeights();
↦ 3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3,3,2,2,1,3,3,2,3,3\
   ,3,2,2,1,3,3,2,3,-22
kill RR, Q, R;
/////////////
// example 3.14 from the paper
intmat Q[3][5] =
1,1,1,1,1,
1,-1,0,0,1,
1,1,1,0,0;
list TOR = 2;
ring R = 0,T(1..5),dp;
// attach degree matrix Q to R:
setBaseMultigrading(Q);
ideal I = T(1)*T(2) + T(3)^2 + T(4)^2;
list TOR = 2;
↦ // ** redefining TOR (list TOR = 2;) ./examples/stabilizer.sing:27
def RR = stabilizer(I, TOR);
↦ // coefficients: QQ
↦ // number of vars : 30
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25)
↦ //        block   2 : ordering C
↦ // ** redefining adMons (  list adMons;) autgradalg.lib::autKS:2324
↦ // ** redefining MONexported (  export(MONexported);)
↦ // ** redefining k (int k = 1;) autgradalg.lib::stabilizer:1975
setring RR;
RR;
↦ // coefficients: QQ
↦ // number of vars : 26
↦ //        block   1 : ordering dp
↦ //                  : names    Y(1) Y(2) Y(3) Y(4) Y(5) Y(6) Y(7) Y(8) Y(\
   9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(17) Y(18) Y(19) Y(20) Y(21\
   ) Y(22) Y(23) Y(24) Y(25)
↦ //        block   2 : ordering dp
↦ //                  : names    Z
↦ //        block   3 : ordering C
Jexported;
↦ Jexported[1]=Y(24)
↦ Jexported[2]=Y(23)
↦ Jexported[3]=Y(22)
↦ Jexported[4]=Y(21)
↦ Jexported[5]=Y(20)
↦ Jexported[6]=Y(17)
↦ Jexported[7]=Y(16)
↦ Jexported[8]=Y(15)
↦ Jexported[9]=Y(12)
↦ Jexported[10]=Y(11)
↦ Jexported[11]=Y(10)
↦ Jexported[12]=Y(9)
↦ Jexported[13]=Y(8)
↦ Jexported[14]=Y(6)
↦ Jexported[15]=Y(5)
↦ Jexported[16]=Y(4)
↦ Jexported[17]=Y(3)
↦ Jexported[18]=Y(2)
↦ Jexported[19]=Y(18)*Y(19)
↦ Jexported[20]=Y(14)*Y(19)
↦ Jexported[21]=Y(13)*Y(18)
↦ Jexported[22]=Y(14)^2-Y(18)^2
↦ Jexported[23]=Y(13)*Y(14)
↦ Jexported[24]=Y(13)^2-Y(19)^2
↦ Jexported[25]=Y(1)*Y(7)-Y(18)^2-Y(19)^2
↦ Jexported[26]=Y(14)*Y(18)^3*Y(25)*Z-Y(13)*Y(19)^3*Y(25)*Z+1
↦ Jexported[27]=Y(19)^5*Y(25)*Z-Y(13)
↦ Jexported[28]=Y(13)*Y(19)^4*Y(25)*Z-Y(19)
↦ Jexported[29]=Y(18)^5*Y(25)*Z+Y(14)
dim(std(Jexported));
↦ 3
stabExported;
↦ [1]:
↦    [1]:
↦       _[1,1]=Y(1)
↦       _[1,2]=0
↦          _[1,3]=0
↦          _[1,4]=0
↦          _[1,5]=0
↦          _[2,1]=0
↦          _[2,2]=Y(7)
↦          _[2,3]=0
↦          _[2,4]=0
↦          _[2,5]=0
↦          _[3,1]=0
↦          _[3,2]=0
↦          _[3,3]=Y(13)
↦          _[3,4]=0
↦          _[3,5]=0
↦          _[4,1]=0
↦          _[4,2]=0
↦          _[4,3]=0
↦          _[4,4]=Y(19)
↦          _[4,5]=0
↦          _[5,1]=0
↦          _[5,2]=0
↦          _[5,3]=0
↦          _[5,4]=0
↦          _[5,5]=Y(25)
↦       [2]:
↦          1,0,0,
↦          0,1,0,
↦          0,0,1
↦       [3]:
↦          _[1]=Y(2)
↦          _[2]=Y(3)
↦          _[3]=Y(4)
↦          _[4]=Y(5)
↦          _[5]=Y(6)
↦          _[6]=Y(8)
↦          _[7]=Y(9)
↦          _[8]=Y(10)
↦          _[9]=Y(11)
↦          _[10]=Y(12)
↦          _[11]=Y(14)
↦          _[12]=Y(15)
↦          _[13]=Y(16)
↦          _[14]=Y(17)
↦          _[15]=Y(18)
↦          _[16]=Y(20)
↦          _[17]=Y(21)
↦          _[18]=Y(22)
↦          _[19]=Y(23)
↦          _[20]=Y(24)
↦          _[21]=Y(1)*Y(7)*Y(13)*Y(19)*Y(25)*Z-1
↦          _[22]=Y(13)^2-Y(19)^2
↦          _[23]=Y(1)*Y(7)-Y(13)^2
↦    [2]:
↦       [1]:
```

```
↦          _[1,1]=Y(1)
↦          _[1,2]=0
↦          _[1,3]=0
↦          _[1,4]=0
↦          _[1,5]=0
↦          _[2,1]=0
↦          _[2,2]=Y(7)
↦          _[2,3]=0
↦          _[2,4]=0
↦          _[2,5]=0
↦          _[3,1]=0
↦          _[3,2]=0
↦          _[3,3]=0
↦          _[3,4]=Y(14)
↦          _[3,5]=0
↦          _[4,1]=0
↦          _[4,2]=0
↦          _[4,3]=Y(18)
↦          _[4,4]=0
↦          _[4,5]=0
↦          _[5,1]=0
↦          _[5,2]=0
↦          _[5,3]=0
↦          _[5,4]=0
↦          _[5,5]=Y(25)
↦       [2]:
↦          1,0,0,
↦          0,1,0,
↦          1,1,1
↦       [3]:
↦          _[1]=Y(2)
↦          _[2]=Y(3)
↦          _[3]=Y(4)
↦          _[4]=Y(5)
↦          _[5]=Y(6)
↦          _[6]=Y(8)
↦          _[7]=Y(9)
↦          _[8]=Y(10)
↦          _[9]=Y(11)
↦          _[10]=Y(12)
↦          _[11]=Y(13)
↦          _[12]=Y(15)
↦          _[13]=Y(16)
↦          _[14]=Y(17)
↦          _[15]=Y(19)
↦          _[16]=Y(20)
↦          _[17]=Y(21)
↦          _[18]=Y(22)
↦          _[19]=Y(23)
↦          _[20]=Y(24)
↦          _[21]=-Y(1)*Y(7)*Y(14)*Y(18)*Y(25)*Z-1
↦          _[22]=-Y(14)^2+Y(18)^2
↦          _[23]=Y(1)*Y(7)-Y(18)^2
```

```
kill RR, Q, R;
```

D.15.1.5 autXhat

Procedure from library `autgradalg.lib` (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: autXhat(RL, w0, TOR): RL is an ideal, w an intvec, TOR a list of integers

Assume: the basering is multigraded, the elements of TOR stand for the torsion rows of the matrix getVariableWeights(), w is an ample class or the free part of such a class.

Purpose: compute an ideal J such that V(J) in some GL(n) is isomorphic to the H-equivariant automorphisms \widehat X –> \widehat X.

Example:

```
LIB "autgradalg.lib";
intmat Q[3][5] =
1,1,1,1,1,
1,-1,0,0,1,
1,1,1,0,0;
list TOR = 2;
ring R = 0,T(1..5),dp;
setBaseMultigrading(Q);
ideal I = T(1)*T(2) + T(3)^2 + T(4)^2;
intvec w0 = 2,1,0;
def RR = autXhat(I, w0, TOR);
↦ // coefficients: QQ
↦ // number of vars : 30
↦ //        block   1 : ordering dp
↦ //                  : names    T(1) T(2) T(3) T(4) T(5) Y(1) Y(2) Y(3) Y(\
   4) Y(5) Y(6) Y(7) Y(8) Y(9) Y(10) Y(11) Y(12) Y(13) Y(14) Y(15) Y(16) Y(1\
   7) Y(18) Y(19) Y(20) Y(21) Y(22) Y(23) Y(24) Y(25)
↦ //        block   2 : ordering C
↦ // ** redefining adMons (  list adMons;) autgradalg.lib::autKS:2324
↦ // ** redefining k (int k = 1;) autgradalg.lib::stabilizer:1975
setring RR;
RES;
↦ [1]:
↦    [1]:
↦       _[1,1]=Y(1)
↦       _[1,2]=0
↦       _[1,3]=0
↦       _[1,4]=0
↦       _[1,5]=0
↦       _[2,1]=0
↦       _[2,2]=Y(7)
↦       _[2,3]=0
↦       _[2,4]=0
↦       _[2,5]=0
↦       _[3,1]=0
↦       _[3,2]=0
↦       _[3,3]=Y(13)
↦       _[3,4]=0
↦       _[3,5]=0
↦       _[4,1]=0
↦       _[4,2]=0
↦       _[4,3]=0
↦       _[4,4]=Y(19)
↦       _[4,5]=0
↦       _[5,1]=0
↦       _[5,2]=0
↦       _[5,3]=0
↦       _[5,4]=0
↦       _[5,5]=Y(25)
↦    [2]:
↦       1,0,0,
↦       0,1,0,
↦       0,0,1
↦    [3]:
↦       _[1]=Y(2)
↦       _[2]=Y(3)
↦       _[3]=Y(4)
↦       _[4]=Y(5)
↦       _[5]=Y(6)
↦       _[6]=Y(8)
↦       _[7]=Y(9)
↦       _[8]=Y(10)
↦       _[9]=Y(11)
↦       _[10]=Y(12)
↦       _[11]=Y(14)
↦       _[12]=Y(15)
↦       _[13]=Y(16)
↦       _[14]=Y(17)
↦       _[15]=Y(18)
↦       _[16]=Y(20)
↦       _[17]=Y(21)
↦       _[18]=Y(22)
↦       _[19]=Y(23)
↦       _[20]=Y(24)
↦       _[21]=Y(1)*Y(7)*Y(13)*Y(19)*Y(25)*Z-1
↦       _[22]=Y(13)^2-Y(19)^2
↦       _[23]=Y(1)*Y(7)-Y(13)^2
↦ [2]:
↦    [1]:
↦       _[1,1]=Y(1)
↦       _[1,2]=0
↦       _[1,3]=0
↦       _[1,4]=0
↦       _[1,5]=0
↦       _[2,1]=0
↦       _[2,2]=Y(7)
↦       _[2,3]=0
↦       _[2,4]=0
↦       _[2,5]=0
↦       _[3,1]=0
↦       _[3,2]=0
↦       _[3,3]=0
↦       _[3,4]=Y(14)
↦        _[3,5]=0
↦        _[4,1]=0
↦        _[4,2]=0
↦        _[4,3]=Y(18)
↦        _[4,4]=0
↦        _[4,5]=0
↦        _[5,1]=0
↦        _[5,2]=0
↦        _[5,3]=0
↦        _[5,4]=0
↦        _[5,5]=Y(25)
↦     [2]:
↦        1,0,0,
↦        0,1,0,
↦        1,1,1
↦     [3]:
↦        _[1]=Y(2)
↦        _[2]=Y(3)
↦        _[3]=Y(4)
↦        _[4]=Y(5)
↦        _[5]=Y(6)
↦        _[6]=Y(8)
↦        _[7]=Y(9)
↦        _[8]=Y(10)
↦        _[9]=Y(11)
↦        _[10]=Y(12)
↦        _[11]=Y(13)
↦        _[12]=Y(15)
↦        _[13]=Y(16)
↦        _[14]=Y(17)
↦        _[15]=Y(19)
↦        _[16]=Y(20)
↦        _[17]=Y(21)
↦        _[18]=Y(22)
↦        _[19]=Y(23)
↦        _[20]=Y(24)
↦        _[21]=-Y(1)*Y(7)*Y(14)*Y(18)*Y(25)*Z-1
↦        _[22]=-Y(14)^2+Y(18)^2
↦        _[23]=Y(1)*Y(7)-Y(18)^2
kill RR, Q, R;
```

D.15.1.6 autX

Procedure from library `autgradalg.lib` (see Section D.15.1 [autgradalg_lib], page 2274).

Usage: autX(RL, w, TOR); RL: ideal, w: intvec, TOR: optional list of integers.

Purpose: compute generators for the hopf algebra O(Aut(X))
of the Mori dream space X given by Cox(X) := basering/RL and the ample class w.

Assume: there is no torsion.

Return: a ring. Also exports an ideal Iexported.

Example:

```
LIB "autgradalg.lib";
///////////////
//// CAREFUL: the following examples seems to be unfeasible at the moment, see remark
//echo=2;
///////////////
//// PP2
//intmat Q[1][4] =
//  1,1,1,1;
//ring R = 0,T(1..ncols(Q)),dp;
//// attach degree matrix Q to R:
//setBaseMultigrading(Q);
//ideal I = 0;
//intvec w0 = 1;
//def RR = autX(I, w0);
//setring RR;
//Iexported;
//basering;
//dim(std(Iexported));
//kill RR, Q, R;
///////////////
//// example 3.14 from the paper
//intmat Q[3][5] =
//  1,1,1,1,1,
//  1,-1,0,0,1,
//  1,1,1,0,0;
//list TOR = 2;
//ring R = 0,T(1..5),dp;
//// attach degree matrix Q to R:
//setBaseMultigrading(Q);
//ideal I = T(1)*T(2) + T(3)^2 + T(4)^2;
//list TOR = 2;
//intvec w0 = 2,1,0;
//def RR = autX(I, w0, TOR);
//setring RR;
//kill RR, Q, R;
```

D.15.2 difform_lib

Library: difform.lib

Purpose: Procedures for differential forms

Author: Peter Chini, chini@rhrk.uni-kl.de

Overview: A library for computing with elements of the differential algebra over a (quotient) ring. To compute in this algebra, a non-commutative ring with additional variables dx_1,...,dx_n and 'exterior' relations between this variables is used. In the case of a quotient ring, the defining ideal and its image under the universal derivation are added as relations. The differential forms themselves are defined via an additional type 'difform'. Objects of this type carry as an attribute a polynomial in the differential algebra and make it available over the basering.

Additionally, the universal derivation is available as a procedure and the differentials between the graded parts of the differential algebra can be applied to differential forms. The library also supports derivations: maps from the first graded part of the differential algebra to the basering. These are defined via the type 'derivation' and there are procedures for basic arithmetic operations, evaluation and Lie-derivative.

Procedures:

D.15.2.1 diffAlgebra

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Side effects:
If R is the basering, the differential algebra is constructed with name Omega_R and the differential forms dx_1,...,dx_n are available. The name of the differential algebra is stored in the attribute attrib(R,"diffAlgebra").

Note: - computations with differential forms need the structure of the differential algebra, so this procedure should be executed first.
- the variable names 'd' or 'D' should be avoided.
- the procedure also works for quotient rings

Example:

```
LIB "difform.lib";
///////////////////////////////////////////////////////////////
// Example for a differential algebra over a polynomial ring //
///////////////////////////////////////////////////////////////
ring R = 0,(a,b,c),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDa, dDb, dDc, da, db, dc are available.
setring Omega_R;
// The differential algebra is given by:
basering;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    Da Db Dc
↦ //        block   2 : ordering ds
↦ //                  : names    a b c
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    DbDa=-Da*Db
↦ //    DcDa=-Da*Dc
↦ //    DcDb=-Db*Dc
↦ // quotient ring from ideal
↦ _[1]=Da^2
↦ _[2]=Db^2
↦ _[3]=Dc^2
kill R,Omega_R,da,db,dc;
/////////////////////////////////////////////////////////////
// Example for a differential algebra over a quotient ring //
/////////////////////////////////////////////////////////////
ring R = 0,(x,y,z),lp;
ideal I = x+y+z,xyz;
qring S = std(I);
diffAlgebra();
```

```
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
setring Omega_R;
// The differential algebra is given by:
basering;
↦ // coefficients: QQ
↦ // number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    Dx Dy Dz
↦ //        block   2 : ordering lp
↦ //                  : names    x y z
↦ //        block   3 : ordering C
↦ // noncommutative relations:
↦ //    DyDx=-Dx*Dy
↦ //    DzDx=-Dx*Dz
↦ //    DzDy=-Dy*Dz
↦ // quotient ring from ideal
↦ _[1]=y^2*z+y*z^2
↦ _[2]=x+y+z
↦ _[3]=Dz*y^4+3*Dz*y^3*z+2*Dz*y^2*z^2
↦ _[4]=Dy*z^3+2*Dz*y^3+5*Dz*y^2*z+2*Dz*y*z^2
↦ _[5]=2*Dy*y*z+Dy*z^2+Dz*y^2+2*Dz*y*z
↦ _[6]=Dx+Dy+Dz
↦ _[7]=Dy*Dz*y^2+2*Dy*Dz*y*z
↦ _[8]=Dz^2
↦ _[9]=Dy^2
↦ _[10]=Dx^2
kill Omega_R,dx,dy,dz;
```

See also: Section D.15.2.3 [diffAlgebraGens], page 2293; Section D.15.2.2 [diffAlgebraStructure], page 2293; Section D.15.2.4 [diffAlgebraUnivDerIdeal], page 2294.

D.15.2.2 diffAlgebraStructure

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diffAlgebraStructure();

Return: the structure of the differential algebra

Remarks: The differential algebra is constructed as non-commutative ring with additional variables Dx_1,...,Dx_n and 'exterior' relations between them. In the case, that the basering is a quotient ring, the defining ideal and its image under the universal derivation are added as relations.

Note: the monomial ordering of the basering is preserved in the differential algebra

See also: Section D.15.2.1 [diffAlgebra], page 2292; Section D.15.2.3 [diffAlgebraGens], page 2293; Section D.15.2.4 [diffAlgebraUnivDerIdeal], page 2294.

D.15.2.3 diffAlgebraGens

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diffAlgebraGens();

Side effects:

The differential forms dx_1,...,dx_n are constructed.

See also: Section D.15.2.1 [diffAlgebra], page 2292; Section D.15.2.2 [diffAlgebraStructure], page 2293; Section D.15.2.4 [diffAlgebraUnivDerIdeal], page 2294.

D.15.2.4 diffAlgebraUnivDerIdeal

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diffAlgebraUnivDerIdeal(I); I ideal

Assume: current basering is the differential algebra of a polynomial ring and I is lifted from the polynomial ring

Return: the image of I under the universal derivation

Remark: The procedure computes the universal derivation of every generator of the ideal.

Note: for differential forms use the procedure difformUnivDer or difformDiff

See also: Section D.15.2.1 [diffAlgebra], page 2292; Section D.15.2.3 [diffAlgebraGens], page 2293; Section D.15.2.2 [diffAlgebraStructure], page 2293.

D.15.2.5 diffAlgebraChangeOrd

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diffAlgebraChangeOrd(#); # list

Assume: the current basering is the differential algebra

Return: the differential algebra with changed monomial ordering

Note: - an ordering is defined via the following pattern:
- #[i] = 'gen' defines the ordering on the generators dx_i - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector
- #[i] = 'ringvar' defines the ordering on the ringvariables - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector
- only use for interior computations
- differential forms are polynomials in the differential algebra - not in the returned ring - do not define differential forms as polynomials in the returned ring since this is another data-ring - an error occurs if: - no valid monomial ordering is given - no weight vector is given but a weighted monomial ordering - a given weight vector has wrong dimension
- weight vectors are ignored if the given ordering is not weighted

See also: Section D.15.2.21 [difformIsBigger], page 2305; Section D.15.2.22 [difformIsSmaller], page 2306; Section D.15.2.27 [difformListSort], page 2310.

D.15.2.6 diffAlgebraListGen

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diffAlgebraListGen(#); # list

Return: - a list of generators of the differential algebra as module over the basering - a list of generators of a graded part of the differential algebra

Remarks: In order to find all generators, they are counted 'binary': The generators are in 1:1 - correspondence to the dual number representations of 1 up to (2^n-1)

Note: - if all generators of the differential algebra are needed, apply the procedure without input
- if the generator(s) of a graded part are needed, apply the procedure with an integer which specifies the wanted degree
- the list of generators is sorted with respect to the monomial ordering on the differential algebra

Example:

```
LIB "difform.lib";
ring R = 11,(x,y,z),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
///////////////////////////////////////////
// Generators of the differential algebra //
///////////////////////////////////////////
diffAlgebraListGen();
↦ [1]:
↦    1
↦
↦ [2]:
↦    dz
↦
↦ [3]:
↦    dy
↦
↦ [4]:
↦    dx
↦
↦ [5]:
↦    dy*dz
↦
↦ [6]:
↦    dx*dz
↦
↦ [7]:
↦    dx*dy
↦
↦ [8]:
↦    dx*dy*dz
↦
//////////////////////////////////////////
// Generators of the second graded part //
//////////////////////////////////////////
diffAlgebraListGen(2);
↦ [1]:
↦    dy*dz
↦
↦ [2]:
↦    dx*dz
↦
↦ [3]:
↦    dx*dy
```

```
↦
kill Omega_R,dx,dy,dz;
```

D.15.2.7 difformFromPoly

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difform df = f; f poly

Return: the differential form of degree 0 defined by f

Remark: The given polynomial gets lifted to the differential algebra and the differential form is defined there.

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
////////////////////////////////////////
// Construction of differential forms //
////////////////////////////////////////
poly f = 3x3z*(y4-y5) + 2;
difform df = f;
df;
↦ 2+3x3y4z-3x3y5z
↦
// For the construction of more general differential forms,
// the constructor difformFromPoly is used implicitly:
difform dg = 3*x*dx - y*dy + dx*dy*dz + 1;
dg;
↦ 1+3x*dx+(-y)*dy+dx*dy*dz
↦
kill Omega_R,df,dg,dx,dy,dz;
```

D.15.2.8 difformCoef

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformCoef(df); df difform

Return: list of lists of differential forms and polynomials:
- the first entry is a generator of the differential algebra which appears in df - the second entry is the corresponding coefficient

Remarks: Via the procedure coef, the coefficients are found - therefore the ring has to be changed to the differential algebra. After that, the coefficients have to be mapped back to the original ring.

Note: the returned list can be sorted with the procedure difformListSort and the optional string 'Llist'

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
```

```
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
difform df = 3*x25*dx - y*dx*dy + 12*dx*dy*dz - dz*dy + 3 + 12*x*dx + 24*(y4-y5) + d
///////////////////////////////
// Unsorted Coefficient List //
///////////////////////////////
list L_1 = difformCoef(df);
L_1;
↦ [1]:
↦    [1]:
↦       dx*dy*dz
↦ 
↦    [2]:
↦       x3+12
↦ [2]:
↦    [1]:
↦       dx*dy
↦ 
↦    [2]:
↦       -y+1
↦ [3]:
↦    [1]:
↦       dy*dz
↦ 
↦    [2]:
↦       1
↦ [4]:
↦    [1]:
↦       dx
↦ 
↦    [2]:
↦       3x25+12x
↦ [5]:
↦    [1]:
↦       dy
↦ 
↦    [2]:
↦       yz5
↦ [6]:
↦    [1]:
↦       dz
↦ 
↦    [2]:
↦       x2+1
↦ [7]:
↦    [1]:
↦       1
↦ 
↦    [2]:
↦       -24y5+24y4+3
/////////////////////////////
// Sorted Coefficient List //
/////////////////////////////
```

```
L_1 = difformListSort(L_1,"Llist","gen","ds");
L_1;
↦ [1]:
↦    [1]:
↦       dx*dy*dz
↦
↦    [2]:
↦       x3+12
↦ [2]:
↦    [1]:
↦       dy*dz
↦
↦    [2]:
↦       1
↦ [3]:
↦    [1]:
↦       dx*dy
↦
↦    [2]:
↦       -y+1
↦ [4]:
↦    [1]:
↦       dz
↦
↦    [2]:
↦       x2+1
↦ [5]:
↦    [1]:
↦       dy
↦
↦    [2]:
↦       yz5
↦ [6]:
↦    [1]:
↦       dx
↦
↦    [2]:
↦       3x25+12x
↦ [7]:
↦    [1]:
↦       1
↦
↦    [2]:
↦       -24y5+24y4+3
kill Omega_R,df,dx,dy,dz,L_1;
```

D.15.2.9 difformGenToString

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformGenToString(df); df difform

Return: the differential form df as a string - but with unsorted coefficients

Note: - this is only used to print generators
- the procedure replaces the 'D' from the variables of the differential algebra by an 'd'

See also: Section D.15.2.11 [difformToString], page 2299.

D.15.2.10 difformHomogDecomp

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformHomogDecomp(df); df difform

Return: list of differential forms: homogeneous decomposition

Note: the output list always has as length the maximal possible degree plus one and the degree-0 part is the last element in the list

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
difform df = 3*dx*dz - x8*dx*dy + 12 + dy*dz + dz*dx - (y4-y5)*x12*dx*dy*dz - dx - dy
///////////////////////////////
// Homogeneous decomposition //
///////////////////////////////
list L = difformHomogDecomp(df);
L;
↦ [1]:
↦    (-1)*dx+(-1)*dy+dz
↦
↦ [2]:
↦    (x2-x8)*dx*dy+2*dx*dz+dy*dz
↦
↦ [3]:
↦    (-x12y4+x12y5)*dx*dy*dz
↦
↦ [4]:
↦    12
↦
kill Omega_R,df,L,dx,dy,dz;
```

See also: Section D.15.2.8 [difformCoef], page 2296; Section D.15.2.23 [difformDeg], page 2307.

D.15.2.11 difformToString

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformToString(df,#); df difform,# list

Return: df as a string, sorted by a given ordering on the generators (standard: the ordering chosen for the differential algebra)

Remarks: The differential form is decomposed via difformCoef, the coefficient list is sorted and then the string is built as concatenation of coefficients and generators

Note: to get a string, respecting a certain monomial ordering on the generators, use: - #[1] = 'gen'
- #[2]: a monomial ordering as string
- #[3]: an optional weight vector

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z,a,b),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dDa, dDb, dx, dy, dz, da, db are available.
difform df = 3*x*dx -2*db + 24*a*dy - y*dx*dy*db + 12*dx*dy*dz - dz*dy*da*db + 3 + 12
//////////////////////////////////////////////////////////////////////////////////
// String sorted with respect to the monomial order on the differential algebra //
//////////////////////////////////////////////////////////////////////////////////
string df_str = difformToString(df);
print(df_str);
↦ dy*dz*da*db+12*dx*dy*dz+(-y)*dx*dy*db+1/77*dx*da+15x*dx+24a*dy+(-2)*db+3
/////////////////////////////////////////////////////////
// String sorted with respect to the weighted order wp //
/////////////////////////////////////////////////////////
df_str = difformToString(df,"gen","wp",intvec(-1,-1,-1,1,1));
print(df_str);
↦ (-2)*db+dy*dz*da*db+1/77*dx*da+3+(-y)*dx*dy*db+15x*dx+24a*dy+12*dx*dy*dz
kill Omega_R,df,df_str,dx,dy,dz,da,db;
```

See also: Section D.15.2.9 [difformGenToString], page 2298; Section D.15.2.12 [difformPrint], page 2300.

D.15.2.12 difformPrint

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df; df difform

Side effects:
prints the differential form

Remarks: Uses the procedure difformToString with a ds-ordering on the generators

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
//////////////////////////
// Application of Print //
//////////////////////////
difform df = 3*x*dx - y*dx*dy + 12*dx*dy*dz - dz*dy + 3 + 12*x*dx;
df;
↦ 3+15x*dx+(-y)*dx*dy+dy*dz+12*dx*dy*dz
↦
kill Omega_R,df,dx,dy,dz;
```

See also: Section D.15.2.9 [difformGenToString], page 2298; Section D.15.2.11 [difformToString], page 2299.

D.15.2.13 difformIsGen

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformIsGen(df); df difform

Return: 1, if df is a generator of the differential algebra - 0, otherwise

Remarks: Uses the procedure difformCoef and tests for a single coefficient which is one

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z,a,b,c),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dDa, dDb, dDc, dx, dy, dz, da, db, dc are available.
////////////////
// Generators //
////////////////
difformIsGen(1);
↦ 1
difformIsGen(dx);
↦ 1
difformIsGen(da*dc);
↦ 1
difformIsGen(dy*da*db);
↦ 1
difformIsGen(-da*dz);
↦ 1
///////////////////
// No generators //
///////////////////
difformIsGen(-1);
↦ 0
difformIsGen(-dx);
↦ 0
difformIsGen(dc*da);
↦ 0
difformIsGen(dy*db*da);
↦ 0
difformIsGen(dx*dz*dy);
↦ 0
kill Omega_R,dx,dy,dz,da,db,dc;
```

See also: Section D.15.2.8 [difformCoef], page 2296.

D.15.2.14 difformAdd

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df+dg; df,dg difform

Return: the sum of the differential forms as differential form

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
```

```
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
////////////////////////////////////
// Addition of differential forms //
////////////////////////////////////
difform df = dx + 4*dy - dz*dx + 4 + 3*dx*dy + 4*dz;
difform dg = dx + dy + 27*dz*dy;
df+dg;
↦ 4+2*dx+5*dy+4*dz+3*dx*dy+dx*dz+(-27)*dy*dz
↦
////////////////////////////////////////////////////
// Addition of polynomials and differential forms //
////////////////////////////////////////////////////
df + x2y2z2;
↦ 4+x2y2z2+dx+4*dy+4*dz+3*dx*dy+dx*dz
↦
12 + dg;
↦ 12+dx+dy+(-27)*dy*dz
↦
kill Omega_R,df,dg,dx,dy,dz;
```

See also: Section D.15.2.15 [difformSub], page 2302.

D.15.2.15 difformSub

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df-dg; df,dg difform

Return: the difference of the differential forms as differential form

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
///////////////////////////////////////
// Subtraction of differential forms //
///////////////////////////////////////
difform df = 5*dx*x2 - 7*dy*z - 2x2*dz - 3;
difform dg = dx - 8x*dz*dy;
df-dg;
↦ -3+(-1+5x2)*dx+(-7z)*dy+(-2x2)*dz+(-8x)*dy*dz
↦
///////////////////////////////////////////////////////
// Subtraction of polynomials and differential forms //
///////////////////////////////////////////////////////
df - 2x3;
↦ -3-2x3+5x2*dx+(-7z)*dy+(-2x2)*dz
↦
1 - dg;
↦ 1+(-1)*dx+(-8x)*dy*dz
↦
```

```
kill Omega_R,df,dg,dx,dy,dz;
```

See also: Section D.15.2.14 [difformAdd], page 2301; Section D.15.2.16 [difformNeg], page 2303.

D.15.2.16 difformNeg

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: -df; df difform

Return: the negation of the differential form

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////////
// Negation of a differential form //
/////////////////////////////////////
difform df = 13*dx*dy + 2*dy*dz - 6*dx*dy*dz - 3;
-df;
↦ 3+(-13)*dx*dy+(-2)*dy*dz+6*dx*dy*dz
↦
kill Omega_R,df,dx,dy,dz;
```

See also: Section D.15.2.15 [difformSub], page 2302.

D.15.2.17 difformMul

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df*dg; df,dg difform

Return: the product of the differential forms as differential form

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////////////
// Multiplication of differential forms //
/////////////////////////////////////////
difform df = 13*dx*dy - 7*dy*dz - 6*dx*dy*dz;
difform dg = dx - 8x;
df*dg;
↦ (-104x)*dx*dy+56x*dy*dz+(-7+48x)*dx*dy*dz
↦
/////////////////////////////////////////////////////////
// Multiplication of polynomials and differential forms //
/////////////////////////////////////////////////////////
df*(y2-x);
↦ (-13x+13y2)*dx*dy+(7x-7y2)*dy*dz+(6x-6y2)*dx*dy*dz
↦
```

```
12*dg;
↦ -96x+12*dx
↦
kill Omega_R,df,dg,dx,dy,dz;
```

See also: Section D.15.2.18 [difformDiv], page 2304.

D.15.2.18 difformDiv

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df/dg; df,dg difform

Return: the quotient df/dg as differential form

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////
// Divisions without remainder //
/////////////////////////////////
dx / dx;
↦ 1
↦
dx*dy*dz / dz;
↦ dx*dy
↦
(dx*x2 - yx2) / x2;
↦ -y+dx
↦
//////////////////////////////
// Divisions with reaminder //
//////////////////////////////
(dx + dx*dy + 1) / dx;
↦ 1+(-1)*dy
↦
(x2*dx - x*dy) / (dx-dy);
↦ x2
↦
kill Omega_R,dx,dy,dz;
```

See also: Section D.15.2.17 [difformMul], page 2303.

D.15.2.19 difformEqu

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df == dg; df,dg difform

Return: 1, if df and dg are euqal - 0, otherwise

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
```

```
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////////////
// Applications of comparison operator //
/////////////////////////////////////////
difform df = 3*dx - x8*dx*dy;
difform dg = 3 + x8*dy;
df == dg;
↦ 0
dg = dg*dx;
df == dg;
↦ 1
kill Omega_R,df,dg,dx,dy,dz;
```

See also: Section D.15.2.20 [difformNeq], page 2305.

D.15.2.20 difformNeq

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df != dg; df,dg difform

Return: 0, if df and dg are euqal - 1, otherwise

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////////////////////
// Applications of negated comparison operator //
/////////////////////////////////////////////////
difform df = 3*dx - x8*dx*dy;
difform dg = 3 + x8*dy;
df != dg;
↦ 1
dg = dg*dx;
df != dg;
↦ 0
kill Omega_R,df,dg,dx,dy,dz;
```

See also: Section D.15.2.19 [difformEqu], page 2304.

D.15.2.21 difformIsBigger

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df > dg OR difformIsBigger(df,dg,#); df,dg difform, # list

Return: - 1, if df is bigger than dg with respect to the monomial ordering in the differential algebra - 0, otherwise - 1, if df is bigger than dg with respect to a given monomial ordering on the generators/ringvariables - 0, otherwise

Note: the procedure uses diffAlgebraChangeOrd to change the order on the differential algebra, therefore an ordering is defined via the following pattern:

- #[i] = 'gen' defines the ordering on the generators dx_i - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector
- #[i] = 'ringvar' defines the ordering on the ringvariables - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
////////////////////////////
// With standard ordering //
////////////////////////////
dx > dy;
↦ 1
x37*dy > dz;
↦ 1
x*dz > y*x*dy;
↦ 0
x3*dx - y*dx*dz > dx*dy*dz;
↦ 0
///////////////////////////
// With changed ordering //
///////////////////////////
difformIsBigger(dx,dy,"gen","ls");
↦ 0
difformIsBigger(x37*dy,dz,"gen","wp",intvec(1,-1,1));
↦ 0
difformIsBigger(x*dz,y*x*dy,"gen","wp",intvec(1,-1,1),"ringvar","wp",intvec(1,-1,1))
↦ 1
difformIsBigger(x3*dx - y*dx*dz,dx*dy*dz,"gen","wp",intvec(-1,-1,1));
↦ 1
kill Omega_R,dx,dy,dz;
```

See also: Section D.15.2.5 [diffAlgebraChangeOrd], page 2294; Section D.15.2.22 [difformIsSmaller], page 2306.

D.15.2.22 difformIsSmaller

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: df < dg OR difformIsSmaller(df,dg,#); df,dg difform, # list

Return: - 1, if df is smaller than dg with respect to the monomial ordering in the differential algebra - 0, otherwise - 1, if df is smaller than dg with respect to a given monomial ordering on the generators/ringvariables - 0, otherwise

Note: the procedure uses diffAlgebraChangeOrd to change the order on the differential algebra, therefore an ordering is defined via the following pattern:
- #[i] = 'gen' defines the ordering on the generators dx_i - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector
- #[i] = 'ringvar' defines the ordering on the ringvariables - #[i+1] must then be a valid monomial ordering as string - #[i+2] an optional weight vector

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
////////////////////////////
// With standard ordering //
////////////////////////////
dz < dy;
↦ 1
x*dz < y*dz;
↦ 0
y2*z2*dy < x;
↦ 0
dx*dz < dy;
↦ 0
///////////////////////////
// With changed ordering //
///////////////////////////
difformIsSmaller(dz,dy,"gen","ls");
↦ 0
difformIsSmaller(x*dz,y*dz,"ringvar","ls");
↦ 1
difformIsSmaller(y2*z2*dy,x,"gen","wp",intvec(1,-1,1));
↦ 1
difformIsSmaller(dx*dz,dy,"gen","ws",intvec(1,-1,-1));
↦ 1
kill Omega_R,dx,dy,dz;
```

See also: Section D.15.2.5 [diffAlgebraChangeOrd], page 2294; Section D.15.2.21 [difformIsBigger], page 2305.

D.15.2.23 difformDeg

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: deg(df); df difform

Return: degree of df - degree of the highest generator, -1 if df = 0

Note: - the procedure does not check if df is homogeneous
- be careful: difformDeg does not cast polynomials to differential forms. So before applying to a polynomial, a type cast should be done

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////
// Degree computations //
/////////////////////////
deg(3*dx - x8*dx*dy);
↦ 2
```

```
deg(3 + x8*dy);
↦ 1
// When applying homog to a polynomial which is considered
// as a differential form, a type cast has to be done first
deg(x2-y);
↦ 2
difform df = x2-y;
deg(df);
↦ 0
kill Omega_R,df,dx,dy,dz;
```

See also: Section D.15.2.24 [difformIsHomog], page 2308; Section D.15.2.25 [difformIsHomogDeg], page 2309.

D.15.2.24 difformIsHomog

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: homog(df); df difform

Return: 1, if df is homogeneous - 0, otherwise

Note: - the form 0 is homogeneous
- be careful: difformIsHomog does not cast polynomials to differential forms. So before applying to a polynomial, a type cast should be done

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////
// Homogeneous //
/////////////////
homog(3*dx*dz - x8*dx*dy);
↦ 0
homog(12x*dx + dy - (y4-y5)*dz);
↦ 0
/////////////////////
// Not homogeneous //
/////////////////////
homog(3 + x8*dy);
↦ 0
homog(x*dx+dy*dx);
↦ 0
// When applying homog to a polynomial which is considered
// as a differential form, a type cast has to be done first
homog(3x-y2);
↦ 0
difform df = 3x-y2;
homog(df);
↦ 1
kill Omega_R,dx,dy,dz,df;
```

See also: Section D.15.2.23 [difformDeg], page 2307; Section D.15.2.25 [difformIsHomogDeg], page 2309.

D.15.2.25 difformIsHomogDeg

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformIsHomogDeg(df,p); df difform, p int

Return: 1, if df is homogeneous of degree p - 0, otherwise

Note: - 0 is homogeneous of degree -1

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
difform df = 3*dx*dz - x8*dx*dy;
difform dg = 3 + x8*dy;
difform dh = 2;
difform dt = 0;
/////////////////////////////////
// Homogeneous of given degree //
/////////////////////////////////
difformIsHomogDeg(df,2);
↦ 0
difformIsHomogDeg(dh,0);
↦ 1
difformIsHomogDeg(dt,-1);
↦ 1
/////////////////////////////////////
// Not homogeneous of given degree //
/////////////////////////////////////
difformIsHomogDeg(df,1);
↦ 0
difformIsHomogDeg(dg,1);
↦ 0
difformIsHomogDeg(dh,1);
↦ 0
kill Omega_R,df,dg,dh,dt,dx,dy,dz;
```

See also: Section D.15.2.23 [difformDeg], page 2307; Section D.15.2.24 [difformIsHomog], page 2308.

D.15.2.26 difformListCont

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformListCont(L,df); L list, df difform

Return: 1, if df is in the list L - 0, otherwise

Note: lists with arbitrary input are allowed

Example:

```
LIB "difform.lib";
ring R = 17,(a,b,c),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDa, dDb, dDc, da, db, dc are available.
list L = "",3,12*a,da-db,16 + dc, 23*da - 4*db*dc*da, db - 4, "entry", dc - db*da, a
//////////////////////////
// Elements in the list //
//////////////////////////
difformListCont(L,da - db);
↦ 1
difformListCont(L,16 + dc);
↦ 1
difformListCont(L,dc - db*da);
↦ 1
//////////////////////////////
// Elements not in the list //
//////////////////////////////
difformListCont(L,22*da);
↦ 0
difformListCont(L,1);
↦ 0
difformListCont(L,a);
↦ 0
kill Omega_R,L,da,db,dc;
```

See also: Section D.15.2.19 [difformEqu], page 2304.

D.15.2.27 difformListSort

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Return: the sorted list L in ascending order, depending on the optional monomial ordering given

Remarks: Classical insertion sort is used to sort the list

Note: - the procedure uses difformIsBigger to compare list elements, therefore an optional ordering is defined via the pattern in difformIsBigger - the standard ordering is the ordering on the differential algebra - the procedure can also handle special lists of lists by using the optional input "Dlist" or "Llist":
- "Dlist" is used for lists with structure:
L[1] is a list of differential forms
L[2] is a list of polynomials of same size,
The list gets sorted by the elements in L[1]. This is mainly used for the structure of derivations.
- "Llist" allows the structure: L[i] is a list with two entries: L[i][1] is a differential form
L[i][2] is a polynomial
The list gets sorted by the elements L[i][1]. This is used to sort coefficient lists.

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z,t),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dDt, dx, dy, dz, dt are available.
/////////////////////////
// Sortation of a list //
/////////////////////////
list L = dx*x, x2 - y*t, 12, dt*dy*dx*dz;
```

```
// Sort list with standard ordering
difformListSort(L);
↦ [1]:
↦    12
↦
↦ [2]:
↦    x2-yt
↦
↦ [3]:
↦    x*dx
↦
↦ [4]:
↦    dx*dy*dz*dt
↦
// Sort list with changed ordering
difformListSort(L,"gen","ls","ringvar","wp",intvec(-1,1,1,1));
↦ [1]:
↦    dx*dy*dz*dt
↦
↦ [2]:
↦    x*dx
↦
↦ [3]:
↦    12
↦
↦ [4]:
↦    x2-yt
↦
/////////////////////////////////////////////
// Sortation of list with structure "Dlist" //
/////////////////////////////////////////////
list DL;
DL[1] = list(dx,x,t,dt); DL[2] = list(y,t*z,4,x);
// This list has the structure described by "Dlist"
difformListSort(DL,"Dlist","ringvar","ls");
↦ [1]:
↦    [1]:
↦       x
↦
↦    [2]:
↦       t
↦
↦    [3]:
↦       dt
↦
↦    [4]:
↦       dx
↦
↦ [2]:
↦    [1]:
↦       zt
↦    [2]:
↦       4
↦    [3]:
↦       x
↦    [4]:
↦       y
/////////////////////////////////////////////
// Sortation of list with structure "Llist" //
/////////////////////////////////////////////
list LL;
LL[1] = list(dx,x); LL[2] = list(t*dt,y); LL[3] = list(x,2);
// This list has the structure described by "Llist"
difformListSort(LL,"Llist");
↦ [1]:
↦    [1]:
↦       x
↦
↦    [2]:
↦       2
↦ [2]:
↦    [1]:
↦       t*dt
↦
↦    [2]:
↦       y
↦ [3]:
↦    [1]:
↦       dx
↦
↦    [2]:
↦       x
kill Omega_R,dx,dy,dz,dt,L,DL,LL;
```

See also: Section D.15.2.21 [difformIsBigger], page 2305; Section D.15.2.22 [difformIsSmaller], page 2306.

D.15.2.28 difformUnivDer

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformUnivDer(f); f poly

Return: a differential form: the image of the universal derivation applied to f

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////////////////////
// Universal derivation applied to polynomials //
/////////////////////////////////////////////////
difformUnivDer(3x);
↦ 3*dx
↦
difformUnivDer(xyz+x2y2z2);
```

```
↦ (2xy2z2+yz)*dx+(2x2yz2+xz)*dy+(2x2y2z+xy)*dz
↦
difformUnivDer(x+y+z);
↦ dx+dy+dz
↦
kill Omega_R,dx,dy,dz;
```

See also: Section D.15.2.29 [difformDiff], page 2313.

D.15.2.29 difformDiff

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: difformDiff(df); df difform

Return: the image of df under the differential

Remark: To compute the image, the universal derivation is applied to each coefficient and multiplied with the corresponding generator

Note: - the differential is a map Omega_R^(p) -> Omega_R^(p+1) and this procedure applies the differential to all homogeneous parts of df
- this procedure can also be applied to polynomials - in this case it is just the universal derivation

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z,a,b,c),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dDa, dDb, dDc, dx, dy, dz, da, db, dc are available.
/////////////////////////////////////////
// Construction of differential forms //
/////////////////////////////////////////
difform df_1 = y*dx + z*dy + x*dz + a*db + b*dc + c*da;
difform df_2 = -5*c4*dc*dz*dy + 3*dx*dz - 13*a4*da*db + 12*a4*da*db + x8*dx*dy + 12 -
poly f = 3x2y2 - z3*c;
////////////////////////////////////////
// Differential applied to the forms //
////////////////////////////////////////
difformDiff(df_1);
↦ (-1)*dx*dy+dx*dz+(-1)*dy*dz+da*db+(-1)*da*dc+db*dc
↦
difformDiff(df_2);
↦ (-2x)*dx*dy*db+(5x12y4-4x12y3)*dx*dy*dz*db
↦
difformDiff(f);
↦ 6xy2*dx+6x2y*dy+(-3z2c)*dz+(-z3)*dc
↦
// The composition of differentials is the zero-map:
difformDiff(difformDiff(df_1));
↦ 0
↦
kill Omega_R,df_1,df_2,f,dx,dy,dz,da,db,dc;
```

See also: Section D.15.2.28 [difformUnivDer], page 2312.

D.15.2.30 derivationFromList

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: derivation phi = derivationFromList(L); L list

Return: the derivation defined by the list L

Remarks: The structure of L is checked and L is sorted,
then it is set as structure list of phi

Note: the structure of L must follow the rules:
- L[1] is a list of all degree-1 generators: all dx_i must occur once and no other differential forms are allowed. The order of the list is not important - L[2] is the list of images of the dx_i: these must be polynomials Since the map is linear, it is enough to store the images of the dx_i

Example:

```
LIB "difform.lib";
ring R = 11,(u,v,w,x),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDu, dDv, dDw, dDx, du, dv, dw, dx are available.
/////////////////////////////////////
// Construction of structure lists //
/////////////////////////////////////
list L_1;
L_1[1] = list(du,dv,dw,dx);
L_1[2] = list(u,v,w,x);
list L_2;
L_2[1] = list(dx,dw,du,dv);
L_2[2] = list(x2,w2,u2,v-wu);
/////////////////////////////////
// Construction of derivations //
/////////////////////////////////
derivation phi = derivationFromList(L_1); phi;
↦  Omega_R^1 --> R
↦        du |--> u
↦        dv |--> v
↦        dw |--> w
↦        dx |--> x
↦
↦
derivation psi = derivationFromList(L_2); psi;
↦  Omega_R^1 --> R
↦        du |--> u2
↦        dv |--> -uw+v
↦        dw |--> w2
↦        dx |--> x2
↦
↦
kill Omega_R,du,dv,dw,dx,phi,psi,L_1,L_2;
```

See also: Section D.15.2.31 [derivationCheckList], page 2315; Section D.15.2.33 [derivationConstructor], page 2316; Section D.15.2.32 [derivationFromPoly], page 2315.

D.15.2.31 derivationCheckList

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: derivationCheckList(L); L list

Remarks: The procedure checks if a given list has the right form for a derivation and throws an error if this is not the case. In particular: - Only degree-1 generators are allowed in L[1] - this is checked via difformIsGen - Any degree-1 generator must occur once - this is checked via difformListCont

Note: like in derivationFromList, the structure of L must follow the rules: - L[1] is a list of all degree-1 generators: all dx_i must occur once and no other differential forms are allowed. The order of the list is not important - L[2] is the list of images of the dx_i: these must be polynomials

See also: Section D.15.2.30 [derivationFromList], page 2314; Section D.15.2.13 [difformIsGen], page 2301; Section D.15.2.26 [difformListCont], page 2309.

D.15.2.32 derivationFromPoly

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: derivation phi = derivationFromPoly(f); f poly

Return: a derivation which maps any degree-1 generator to f

Remarks: The degree-1 generators are returned by diffAlgebraListGen

Note: the procedure allows to interpret polynomials as derivations

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
//////////////////////////////////////////////////
// Construction of derivations from polynomials //
//////////////////////////////////////////////////
derivation phi = derivationFromPoly(3x*y - 12*y4-z2); phi;
↦  Omega_R^1 --> R
↦        dx |--> 3xy-12y4-z2
↦        dy |--> 3xy-12y4-z2
↦        dz |--> 3xy-12y4-z2
↦
↦
derivation psi = derivationFromPoly(0); psi;
↦  Omega_R^1 --> R
↦        dx |--> 0
↦        dy |--> 0
↦        dz |--> 0
↦
↦
kill Omega_R,dx,dy,dz,phi,psi;
```

See also: Section D.15.2.33 [derivationConstructor], page 2316; Section D.15.2.30 [derivationFromList], page 2314.

D.15.2.33 derivationConstructor

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: derivation phi = inp; inp of any type

Return: the derivation defined by inp:

Remarks: the output depens on the type of inp:
- if inp is of type list, the constructor derivationFromList is used - if inp is of type poly, number, int or bigint, derivationFromPoly is used

Note: for other than the mentioned types, there is no output

Example:

```
LIB "difform.lib";
ring R = 31,(x,y,z),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
////////////////////////////////////////////////////////////
// Construction of derivations from lists and polynomials //
////////////////////////////////////////////////////////////
list L; L[1] = list(dx,dz,dy); L[2] = list(x2,y-x,z);
derivation phi = L; phi;
↦  Omega_R^1 --> R
↦        dx |--> x2
↦        dy |--> z
↦        dz |--> -x+y
↦
↦
derivation psi = 3x2-12z; psi;
↦  Omega_R^1 --> R
↦        dx |--> 3x2-12z
↦        dy |--> 3x2-12z
↦        dz |--> 3x2-12z
↦
↦
kill Omega_R,dx,dy,dz,phi,psi;
```

See also: Section D.15.2.30 [derivationFromList], page 2314; Section D.15.2.32 [derivationFromPoly], page 2315.

D.15.2.34 derivationToString

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: derivationToString(phi,#); phi derivation, # list

Return: the derivation as a string, describing the image of the degree-1 generators, optionally ordered by a given monomial ordering on the generators

Remarks: To sort the images of the generators in the output string, difformListSort is used.

Note: to define an ordering for the generators, one can use:
- #[1] = 'gen'
- #[2]: a monomial ordering as string
- #[3]: an optional weight vector
the standard ordering is the ordering of the differential algebra

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
list L; L[1] = list(dx,dy,dz); L[2] = list(x2,y-23xz,xz4);
derivation phi = L;
///////////////////////////////////
// String with standard ordering //
///////////////////////////////////
print(derivationToString(phi));
↦  Omega_R^1 --> R
↦        dz |--> xz4
↦        dy |--> -23xz+y
↦        dx |--> x2
↦
//////////////////////////////////
// String with changed ordering //
//////////////////////////////////
print(derivationToString(phi,"gen","wp",intvec(-1,-1,1)));
↦  Omega_R^1 --> R
↦        dy |--> -23xz+y
↦        dx |--> x2
↦        dz |--> xz4
↦
kill Omega_R,dx,dy,dz,L,phi;
```

See also: Section D.15.2.35 [derivationPrint], page 2317; Section D.15.2.9 [difformGenToString], page 2298; Section D.15.2.27 [difformListSort], page 2310.

D.15.2.35 derivationPrint

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi; phi derivation

Side effects:
prints the given derivation

Remarks: Prints the string returned by derivationToString with a ls-ordering on the generators

Example:

```
LIB "difform.lib";
ring R = 0,(a,b,c,x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDa, dDb, dDc, dDx, dDy, dDz, da, db, dc, dx, dy, dz are available.
/////////////////
// Derivations //
/////////////////
list L; L[1] = list(dx,dy,dz,da,db,dc); L[2] = list(1,12x-y,z4aby, 2*b5x,0,xyz-abc);
derivation phi = L;
derivation phi_poly = 3ab - c2*x + z;
///////////////////////////
```

```
// Applications of Print //
///////////////////////////
phi;
↦   Omega_R^1 --> R
↦         da |--> 2b5x
↦         db |--> 0
↦         dc |--> -abc+xyz
↦         dx |--> 1
↦         dy |--> 12x-y
↦         dz |--> abyz4
↦
↦
phi_poly;
↦   Omega_R^1 --> R
↦         da |--> 3ab-c2x+z
↦         db |--> 3ab-c2x+z
↦         dc |--> 3ab-c2x+z
↦         dx |--> 3ab-c2x+z
↦         dy |--> 3ab-c2x+z
↦         dz |--> 3ab-c2x+z
↦
↦
kill Omega_R,da,db,dc,dx,dy,dz,L,phi,phi_poly;
```

See also: Section D.15.2.34 [derivationToString], page 2316.

D.15.2.36 derivationAdd

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi+psi; phi,psi derivation

Return: the sum of the given derivations

Remark: The sum is computed componentwise - this works since the structure lists of derivations are sorted the same way.

Note: once can also add polynomials and derivations

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),ds;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
list L_1; L_1[1] = list(dx,dy,dz); L_1[2] = list(2x,2y,2z);
list L_2; L_2[1] = list(dx,dy,dz); L_2[2] = list(y2-x,z4+x+y,y2);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦   Omega_R^1 --> R
↦         dx |--> 2x
↦         dy |--> 2y
↦         dz |--> 2z
↦
```

```
↦
derivation phi_2 = L_2; phi_2;
↦  Omega_R^1 --> R
↦        dx |--> -x+y2
↦        dy |--> x+y+z4
↦        dz |--> y2
↦
↦
////////////////////////
// Sum of derivations //
////////////////////////
phi_1 + phi_2;
↦  Omega_R^1 --> R
↦        dx |--> x+y2
↦        dy |--> x+3y+z4
↦        dz |--> 2z+y2
↦
↦
phi_1 + phi_2 + phi_2;
↦  Omega_R^1 --> R
↦        dx |--> 2y2
↦        dy |--> 2x+4y+2z4
↦        dz |--> 2z+2y2
↦
↦
phi_1 + phi_2 + 3x2;
↦  Omega_R^1 --> R
↦        dx |--> x+3x2+y2
↦        dy |--> x+3y+3x2+z4
↦        dz |--> 2z+3x2+y2
↦
↦
kill Omega_R,dx,dy,dz,L_1,L_2,phi_1,phi_2;
```

See also: Section D.15.2.37 [derivationSub], page 2319.

D.15.2.37 derivationSub

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi-psi; phi,psi derivation

Return: the difference of the given derivations

Remarks: The difference is computed componentwise - this works since the structure lists of derivations are sorted the same way.

Note: one can also subtract polynomials from derivations

Example:

```
LIB "difform.lib";
ring R = 0,(x,y),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dx, dy are available.
list L_1; L_1[1] = list(dx,dy); L_1[2] = list(x+y,1);
list L_2; L_2[1] = list(dy,dx); L_2[2] = list(x,y2);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦   Omega_R^1 --> R
↦         dx |--> x+y
↦         dy |--> 1
↦
↦
derivation phi_2 = L_2; phi_2;
↦   Omega_R^1 --> R
↦         dx |--> y2
↦         dy |--> x
↦
↦
///////////////////////////////
// Difference of derivations //
///////////////////////////////
phi_1-phi_2;
↦   Omega_R^1 --> R
↦         dx |--> x-y2+y
↦         dy |--> -x+1
↦
↦
phi_1-phi_2-phi_1;
↦   Omega_R^1 --> R
↦         dx |--> -y2
↦         dy |--> -x
↦
↦
phi_1 - (x+y);
↦   Omega_R^1 --> R
↦         dx |--> 0
↦         dy |--> -x-y+1
↦
↦
kill Omega_R,dx,dy,L_1,L_2,phi_1,phi_2;
```

See also: Section D.15.2.36 [derivationAdd], page 2318; Section D.15.2.38 [derivationNeg], page 2320.

D.15.2.38 derivationNeg

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: -phi; phi derivation

Return: the negation of a given derivation

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z,t),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dDt, dx, dy, dz, dt are available.
list L_1; L_1[1] = list(dy,dx,dt,dz); L_1[2] = list(x2-y,23y+t,tz4,z-y);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦  Omega_R^1 --> R
↦        dx |--> 23y+t
↦        dy |--> x2-y
↦        dz |--> -y+z
↦        dt |--> z4t
↦
↦
derivation phi_poly = 3xyz; phi_poly;
↦  Omega_R^1 --> R
↦        dx |--> 3xyz
↦        dy |--> 3xyz
↦        dz |--> 3xyz
↦        dt |--> 3xyz
↦
↦
/////////////////////////////
// Negation of derivations //
/////////////////////////////
-phi_1;
↦  Omega_R^1 --> R
↦        dx |--> -23y-t
↦        dy |--> -x2+y
↦        dz |--> y-z
↦        dt |--> -z4t
↦
↦
-(-phi_1);
↦  Omega_R^1 --> R
↦        dx |--> 23y+t
↦        dy |--> x2-y
↦        dz |--> -y+z
↦        dt |--> z4t
↦
↦
-(phi_poly);
↦  Omega_R^1 --> R
↦        dx |--> -3xyz
↦        dy |--> -3xyz
↦        dz |--> -3xyz
↦        dt |--> -3xyz
↦
↦
kill Omega_R,dx,dy,dz,dt,L_1,phi_1,phi_poly;
```

See also: Section D.15.2.37 [derivationSub], page 2319.

D.15.2.39 derivationMul

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi*psi; phi,psi derivation

Return: the componentwise product of phi and psi

Remarks: The product is computed componentwise - this works since the structure lists of derivations are sorted the same way.

Note: one can also multiply polynomials and derivations

Example:

```
LIB "difform.lib";
ring R = 0,(a,b,t),ls;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDa, dDb, dDt, da, db, dt are available.
list L_1; L_1[1] = list(da,dt,db); L_1[2] = list(2a,2t-b,2t);
list L_2; L_2[1] = list(dt,db,da); L_2[2] = list(-a,-b,-t);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦  Omega_R^1 --> R
↦        da |--> 2a
↦        db |--> 2t
↦        dt |--> 2t-b
↦
↦
derivation phi_2 = L_2; phi_2;
↦  Omega_R^1 --> R
↦        da |--> -t
↦        db |--> -b
↦        dt |--> -a
↦
↦
///////////////////////////////////
// Multiplication of derivations //
///////////////////////////////////
phi_1*phi_2;
↦  Omega_R^1 --> R
↦        da |--> -2at
↦        db |--> -2bt
↦        dt |--> -2at+ab
↦
↦
phi_1*phi_2*phi_2;
↦  Omega_R^1 --> R
↦        da |--> 2at2
↦        db |--> 2b2t
↦        dt |--> 2a2t-a2b
↦
↦
```

```
phi_2*(3a2-bt);
↦  Omega_R^1 --> R
↦        da |--> bt2-3a2t
↦        db |--> b2t-3a2b
↦        dt |--> abt-3a3
↦
↦
kill Omega_R,da,db,dt,L_1,L_2,phi_1,phi_2;
```

D.15.2.40 derivationEqu

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi == psi; phi,psi derivation

Return: 1, if phi and psi are equal - 0, otherwise

Remarks: The images of the generators are compared compononentwise - this works since the structure lists of derivations are sorted the same way.

Note: derivations can also be compared to polynomials

Example:

```
LIB "difform.lib";
ring R = 0,(u,v),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDu, dDv, du, dv are available.
list L_1; L_1[1] = list(dv,du); L_1[2] = list(u,-v);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦  Omega_R^1 --> R
↦        du |--> -v
↦        dv |--> u
↦
↦
derivation phi_poly = u*v; phi_poly;
↦  Omega_R^1 --> R
↦        du |--> uv
↦        dv |--> uv
↦
↦
///////////////////////////////
// Comparison of derivations //
///////////////////////////////
phi_1 == phi_1;
↦ 1
phi_1 == phi_poly;
↦ 0
phi_poly == u*v;
↦ 1
kill Omega_R,du,dv,phi_1,phi_poly;
```

See also: Section D.15.2.41 [derivationNeq], page 2324.

D.15.2.41 derivationNeq

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi != psi; phi,psi derivation

Return: 0, if phi and psi are equal - 1, otherwise

Remarks: The comparison is done by difformEqu

Note: derivations can also be compared to polynomials

Example:

```
LIB "difform.lib";
ring R = 0,(u,v),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDu, dDv, du, dv are available.
list L_1; L_1[1] = list(dv,du); L_1[2] = list(u,-v);
/////////////////
// Derivations //
/////////////////
derivation phi_1 = L_1; phi_1;
↦  Omega_R^1 --> R
↦        du |--> -v
↦        dv |--> u
↦
↦
derivation phi_poly = u*v; phi_poly;
↦  Omega_R^1 --> R
↦        du |--> uv
↦        dv |--> uv
↦
↦
///////////////////////////////
// Comparison of derivations //
///////////////////////////////
phi_1 != phi_1;
↦ 0
phi_1 != phi_poly;
↦ 1
phi_poly != u*v;
↦ 0
kill Omega_R,du,dv,phi_1,phi_poly;
```

See also: Section D.15.2.40 [derivationEqu], page 2323.

D.15.2.42 derivationEval

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: phi(df); phi derivation, df difform

Return: the polynomial phi(df), the derivation phi evaluated at df

Remarks: - By linearity it is enough to compute the sum of all differential forms: (coefficient of dx_i)*(image of dx_i)

- The coefficient list of df is computed via difformCoef - To avoid searching generators in lists, the coefficient list of df and the structure list of phi are sorted the same way

Note: - the differential form 0 is allowed as input
- an error will occur if the given differential form is not of degree 1 or -1

Example:

```
LIB "difform.lib";
ring R = 13,(x,y,z,t),dp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
  forms dDx, dDy, dDz, dDt, dx, dy, dz, dt are available.
/////////////////////////////////
// Construction of derivations //
/////////////////////////////////
list L_1; L_1[1] = list(dx,dt,dz,dy); L_1[2] = list(x,y,z,t);
derivation phi_1 = L_1; phi_1;
↦  Omega_R^1 --> R
↦        dx |--> x
↦        dy |--> t
↦        dz |--> z
↦        dt |--> y
↦
↦
list L_2; L_2[1] = list(dx,dy,dz,dt); L_2[2] = list(y2x-zt,zt + y, t3-x, y4-y5);
derivation phi_2 = L_2; phi_2;
↦  Omega_R^1 --> R
↦        dx |--> xy2-zt
↦        dy |--> zt+y
↦        dz |--> t3-x
↦        dt |--> -y5+y4
↦
↦
list L_3; L_3[1] = list(dx,dy,dz,dt); L_3[2] = list(0,0,0,0);
derivation phi_3 = L_3; phi_3;
↦  Omega_R^1 --> R
↦        dx |--> 0
↦        dy |--> 0
↦        dz |--> 0
↦        dt |--> 0
↦
↦
///////////////////////////////
// Evaluation of derivations //
///////////////////////////////
phi_1(0);
↦ 0
phi_1(dx+dy+dz+dt);
↦    ? Cannot apply derivation to non-homogeneous element!
↦    ? leaving difform.lib::derivationEval (0)
phi_1(3*dx - dt);
↦    ? Cannot apply derivation to non-homogeneous element!
↦    ? leaving difform.lib::derivationEval (0)
phi_2(dt);
```

```
↦ -y5+y4
phi_2(dx+dt);
↦    ? Cannot apply derivation to non-homogeneous element!
↦    ? leaving difform.lib::derivationEval (0)
phi_2(dx - dy + (x3-y2)*dz + 12*dt);
↦    ? Cannot apply derivation to non-homogeneous element!
↦    ? leaving difform.lib::derivationEval (0)
phi_3(dx);
↦ 0
phi_3(dy);
↦ 0
phi_3(dx - 24*(dx + dz) - x4*dy);
↦ 0
kill Omega_R,dx,dy,dz,dt,L_1,L_2,L_3,phi_1,phi_2,phi_3;
```

See also: Section D.15.2.8 [difformCoef], page 2296; Section D.15.2.27 [difformListSort], page 2310.

D.15.2.43 derivationContractionGen

Procedure from library difform.lib (see Section D.15.2 [difform_lib], page 2291).

Usage: derivationContractionGen(phi,d_gen); phi derivation, d_gen difform

Assume: d_gen is a generator of the differential algebra

Return: the image of d_gen under the contraction map i_phi

Remarks: The formula for the contraction map applied to a generator of degree l is given by: i_phi^(l)(dx_k*...*dx_j) = sum(i=1,..l)(-1)^(i+1) * phi(dx_i) * (dx_k*...*dx_j / dx_i)

Note: this procedure should only be applied to generators

See also: Section D.15.2.44 [derivationContraction], page 2326.

D.15.2.44 derivationContraction

Procedure from library difform.lib (see Section D.15.2 [difform_lib], page 2291).

Usage: derivationContraction(phi,df); phi derivation, df difform

Return: the image of the contraction map i_phi applied to df

Remarks: Since the contraction map is linear, it is only applied to the generators: So the image of df under i_phi is a sum, where the coefficients are multiplied by the image of the generators.

Note: over the basering, the contraction map is the 0-map

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
   forms dDx, dDy, dDz, dx, dy, dz are available.
/////////////////////////////////
// Construction of derivations //
/////////////////////////////////
list L_1; L_1[1] = list(dx,dy,dz); L_1[2] = list(x,y,z);
derivation phi_1 = L_1; phi_1;
```

```
↦  Omega_R^1 --> R
↦        dx |--> x
↦        dy |--> y
↦        dz |--> z
↦
↦
list L_2; L_2[1] = list(dx,dy,dz); L_2[2] = list(y-x,z-y,x-z);
derivation phi_2 = L_2; phi_2;
↦  Omega_R^1 --> R
↦        dx |--> -x+y
↦        dy |--> -y+z
↦        dz |--> x-z
↦
↦
//////////////////////////////////
// Contractions of derivations //
//////////////////////////////////
derivationContraction(phi_1,dx+dy+dz);
↦ x+y+z
↦
derivationContraction(phi_1,x2*y4-z);
↦ 0
↦
derivationContraction(phi_1,x2*dx*dy + dx*dy*dz);
↦ (-x2y)*dx+(-x3)*dy+z*dx*dy+y*dx*dz+x*dy*dz
↦
derivationContraction(phi_2,dx+dy+dz);
↦ 0
↦
derivationContraction(phi_2,dx*dy*dz - dx*dy + dx*dz);
↦ (-x-y+2z)*dx+(-x+y)*dy+(x-y)*dz+(x-z)*dx*dy+(-y+z)*dx*dz+(-x+y)*dy*dz
↦
kill Omega_R,dx,dy,dz,L_1,L_2,phi_1,phi_2;
```

See also: Section D.15.2.44 [derivationContraction], page 2326; Section D.15.2.45 [derivationLie], page 2327.

D.15.2.45 derivationLie

Procedure from library `difform.lib` (see Section D.15.2 [difform_lib], page 2291).

Usage: diff(phi,df); phi derivation, df difform

Return: the image of df under the Lie-derivative L_phi

Remarks: The map L_phi is the anticommutator of the contraction map i_phi and the differential d:
(i_phi o d) + (d o i_phi)

Example:

```
LIB "difform.lib";
ring R = 0,(x,y,z),lp;
diffAlgebra();
↦ // The differential algebra Omega_R was constructed and the differential \
  forms dDx, dDy, dDz, dx, dy, dz are available.
```

```
/////////////////////////////////
// Construction of derivations //
/////////////////////////////////
list L; L[1] = list(dx,dy,dz); L[2] = list(x2,y2,z2);
derivation phi = L; phi;
↦  Omega_R^1 --> R
↦        dx |--> x2
↦        dy |--> y2
↦        dz |--> z2
↦
↦
derivation phi_poly = x-y;
///////////////////////////////////
// Lie-derivative of derivations //
///////////////////////////////////
diff(phi,dx);
↦ 2x*dx
↦
diff(phi,dx*dy);
↦ (-2x+2y)*dx*dy
↦
diff(phi,dx*dy*dz);
↦ (2x-2y+2z)*dx*dy*dz
↦
diff(phi,dx*dy + dy*dx);
↦ 0
↦
diff(phi,dx*dy - dy*dx);
↦ (-4x+4y)*dx*dy
↦
diff(phi_poly,dx);
↦ dx+(-1)*dy
↦
diff(phi_poly,dx-dy);
↦ 0
↦
diff(phi_poly,dx+dy);
↦ 2*dx+(-2)*dy
↦
diff(phi_poly,dx*(x2-y4) + 1);
↦ (3x2-4xy3-2xy+3y4)*dx+(-x2-8xy3+9y4)*dy
↦
kill Omega_R,dx,dy,dz,L,phi,phi_poly;
```

See also: Section D.15.2.44 [derivationContraction], page 2326; Section D.15.2.29 [difformDiff], page 2313.

D.15.3 enumpoints_lib

Library: enumpoints.lib

Purpose: enumerating rational points

Author: Jieao Song (8d1h)

Procedures:

D.15.3.1 points

Procedure from library `enumpoints.lib` (see Section D.15.3 [enumpoints_lib], page 2328).

Usage: points(I); I homogeneous ideal

Return: list of coordinates of points on the zero set of I

Assumes: dim(I)==0

D.15.3.2 projPoints

Procedure from library `enumpoints.lib` (see Section D.15.3 [enumpoints_lib], page 2328).

Return: list of coordinates of points on the zero set of I

Assumes: dim(I)==0

Example:

```
LIB "enumpoints.lib";
// The set of nodes is a 0-dimensional variety over a number field.
ring R2 = (0,q),(x,y,z,w),dp;
minpoly = q16-q12+q8-q4+1;
poly s = 2q13-q9-q7+q5-q3-q;
ideal Togliatti = 64*(x-w)*(x^4 -4*x^3*w -10*x^2*y^2 -4*x^2*w^2 +16*x*w^3 -20*x*y^2*w
matrix Jac = jacob(Togliatti);
ideal I2 = Togliatti+Jac;
list L=projPoints(std(I2));
L[1];
↦ [1]:
↦    1
↦ [2]:
↦    0
↦ [3]:
↦    (1/2q13-1/4q9-1/4q7+1/4q5-1/4q3-1/4q)
↦ [4]:
↦    (-q12+q8+3/2)
size(L);
↦ 31
```

D.15.4 finitediff_lib

Issues:

installation of qepcadfilter.pl needs to be solved

tests for (nearly) all procedures are missing

global variables needs to be cleaned

temporary files needs to be cleaned

temporary file names need to be unique (think about multiple instances)

pollution of global Top namespace must be solved

u is not a good name for a procedure

Library: finitediff.lib

Purpose: procedures to compute finite difference schemes for linear differential equations

Author: Christian Dingler

Overview: Using qepcad/qepcadsystem from this library requires the program qepcad to be installed. You can download qepcad from http://www.usna.edu/CS/qepcadweb/B/QEPCAD.html

Procedures:

D.15.4.1 visualize

Procedure from library finitediff.lib (see Section D.15.4 [finitediff_lib], page 2329).

Usage: visualize(f); f of type poly.

Return: type string; translates the polynomial form of a finite difference scheme into an indexed one as often seen in literature

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
scheme(u(Ut)+a*u(Ux),trapezoid(Ux,U,x),backward(Ut,U,t));
↦ (dx)*x+(dx)
visualize(_);
↦ (dx)*U(0,1)+(dx)*U(0,0)
```

D.15.4.2 u

Procedure from library finitediff.lib (see Section D.15.4 [finitediff_lib], page 2329).

Usage: u(D[,#]); D a string that occurs in the list of @derivatives, # an optional list of integers.

Return: type vector; gives the vector, that corresponds with gen(n)*m, where m is the monomial defined by #

Example:

```
LIB "finitediff.lib";
list D="Ux","Uy","Ut","U";
list P="a","b";
list V="t","x","y";
setinitials(V,D,P);
u(Ux);
↦ [1]
u(Ux,2,3,7);
↦ [t^2*x^3*y^7]
u(Uy)+u(Ut)-u(Ux);
↦ [-1,1,1]
u(U)*234-dx*dt*dy*3*u(Uy);
↦ [0,(-3*dt*dx*dy),0,234]
```

D.15.4.3 scheme

Procedure from library finitediff.lib (see Section D.15.4 [finitediff_lib], page 2329).

Usage: scheme([v1,..,vn]); v1,..,vn of type vector

Return: poly

Purpose: performs substitutions by the means of Groebner basis computation of the submodule, generated by the input vectors, then intersects the intermediate result with the suitable component in order to get a finite difference scheme

Note: works only for a single PDE, for the case of a system use `matrixsystem`

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
def s1=scheme(u(Ut)+a*u(Ux),backward(Ux,U,x),forward(Ut,U,t));
s1;
↦ (-a*dt+dx)/(dx)*x+(a*dt)/(dx)
```

D.15.4.4 laxfrT

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: laxfrT(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the Lax-Friedrich-approximation for the derivation in the timevariable as often used in literature;

Note: see also laxfrX, setinitials, scheme; Warning: laxfrT is not to be interchanged with laxfrX

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
laxfrT(Ux,U,x);
↦ [(dt)*x,0,-t*x+1/2*x^2+1/2]
```

D.15.4.5 laxfrX

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: laxfrX(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the Lax-Friedrich-approximation for the derivation in one of the spatial variables as often used in literature;

Note: see also laxfrT, setinitials, scheme; Warning: laxfrX is not to be interchanged with laxfrT

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
laxfrX(Ux,U,x);
↦ [(2*dx)*x,0,-x^2+1]
```

D.15.4.6 forward

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: forward(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the forward approximation as often used in literature;

Note: see also laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
forward(Ux,U,x);
↦ [0,(dx),0,-x+1]
forward(Uy,U,y);
↦ [0,0,(dy),-y+1]
forward(Ut,U,t);
↦ [(dt),0,0,-t+1]
```

D.15.4.7 backward

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: backward(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the backward approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
backward(Ux,U,x);
↦ [0,(dx)*x,0,-x+1]
backward(Uy,U,y);
↦ [0,0,(dy)*y,-y+1]
backward(Ut,U,t);
↦ [(dt)*t,0,0,-t+1]
```

D.15.4.8 central1st

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: central1st(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the first-order-central-approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
central1st(Ux,U,x);
↦ [0,(2*dx)*x,0,-x^2+1]
central1st(Uy,U,y);
↦ [0,0,(2*dy)*y,-y^2+1]
```

D.15.4.9 central2nd

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: central2nd(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the second-order-central-approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Uxx","Ux","Utt","Ut","U";
list P="lambda";
list V="t","x";
setinitials(V,D,P);
central2nd(Uxx,U,x);
↦ [(dx^2)*x,0,0,0,-x^2+2*x-1]
central2nd(Utt,U,t);
↦ [0,0,(dt^2)*t,0,-t^2+2*t-1]
```

D.15.4.10 trapezoid

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: trapezoid(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the trapezoid-approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Uxx","Ux","Utt","Ut","U";
list P="lambda";
list V="t","x";
setinitials(V,D,P);
trapezoid(Uxx,Ux,x);
↦ [(dx)/2*x+(dx)/2,-x+1]
trapezoid(Ux,U,x);
↦ [0,(dx)/2*x+(dx)/2,0,0,-x+1]
```

D.15.4.11 midpoint

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: midpoint(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the midpoint-approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Uxx","Ux","Utt","Ut","U";
list P="lambda";
list V="t","x";
setinitials(V,D,P);
midpoint(Ux,U,x);
↦ [0,(2*dx)*x,0,0,-x^2+1]
```

D.15.4.12 pyramid

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: pyramid(U1,U2,var); U1, U2 are the names of occuring derivatives, var is a variable in the basering;

Return: type vector; gives a predefined approximation of the pyramid-approximation as often used in literature;

Note: see also forward,laxfrT,setinitials,scheme;

Example:

```
LIB "finitediff.lib";
list D="Uxx","Ux","Utt","Ut","U";
list P="lambda";
list V="t","x";
setinitials(V,D,P);
pyramid(Ux,U,x);
↦ [0,(dx)/3*x^2+(dx)/3*x+(dx)/3,0,0,-x^3+x]
```

D.15.4.13 setinitials

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: setinitials(V,D[,P]); V,D,P are lists with strings as elements

Return: no return value: sets the dependence order of the occuring derivatives, constructs the suitable ring to compute in containing user chosen parameters, sets new basering

Note: P is optional, used to introduce some additional parameters into the ring. The Sine and Cosine values needed for the fourier transformation are symbolically introduced under the names string(c)+nameof(variable), i.e. if x is any spatial variable then cx:=cosine(dx*ksi), when regarding the fourier transform after ksi (for sine respectively). Artificial parameters I,T,Px,Py are introduced for the later eigenvalue analysis. Variables can be transformed into parameters of similar name

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="alpha","beta","gamma";
setinitials(V,D,P);////does not show the ring, since there is no output
basering;///does show the ring
↦ // coefficients: QQ(I, T, Px, Py, Cx, Cy, Sx, Sy, alpha, beta, gamma, dt,\
   dx, dy)
↦ // number of vars : 8
↦ //        block   1 : ordering c
↦ //        block   2 : ordering lp
↦ //                  : names    i t x y cx cy sx sy
↦ // quotient ring from ideal
↦ _[1]=cy^2+sy^2-1
↦ _[2]=cx^2+sx^2-1
↦ _[3]=i^2+1
```

D.15.4.14 errormap

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: errormap(f); f of type poly

Return: type poly; performs the fouriertransformation of a single polynomial

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
scheme(u(Ut)+a*u(Ux),central1st(Ux,U,x),backward(Ut,U,t));
↦ (2*dx)*x
errormap(_);
↦ (2*dx)*i*sx+(2*dx)*cx
```

D.15.4.15 matrixsystem

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: matrixsystem(M,A); where the M=Mt,M1,..,Mn is a list with square matrices of the same dimension as entries, and A=At,A1,..,An gives the corresponding approximations for the several variables (t,x1,..,xn) as vector. Intended to solve Mt*U_t + M1*U_x1+..+Mn*U_xn=0 as a linear sytem of partial differential equations numerically by a finite difference scheme;

Return: type matrix; gives back the matrices B1,B2 that represent the finite difference scheme, partitioned into different time levels in the form: B1*U(t=N)=B2*U(t<N), where N is the maximal occurring degree (timelevel) of t.

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
list Mat=unitmat(2),unitmat(2);
list Appr=forward(Ut,U,t),forward(Ux,U,x);
matrixsystem(Mat,Appr);
↦ _[1,1]=-1/(dt)*t-1/(dx)*x+(dt+dx)/(dt*dx)
↦ _[1,2]=0
↦ _[2,1]=0
↦ _[2,2]=-1/(dt)*t-1/(dx)*x+(dt+dx)/(dt*dx)
```

D.15.4.16 timestep

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: timestep(M); M a square matrix with polynomials over the basering as entries;

Return: type list; gives two matrices M1,M2 that are the splitting of M with respect to the degree of the variable t in the entries, where the first list-entry M1 consists of the polynomials of the highest timelevel and M2 of the lower levels in the form: M=0 => M1=M2, i.e. M1-M2=M

Note: intended to be used for the finite-difference-scheme-construction and partition into the several time steps

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
list Mat=unitmat(2),unitmat(2);
list Apr=forward(Ut,U,t),forward(Ux,U,x);
matrixsystem(Mat,Apr);
↦ _[1,1]=-1/(dt)*t-1/(dx)*x+(dt+dx)/(dt*dx)
↦ _[1,2]=0
↦ _[2,1]=0
↦ _[2,2]=-1/(dt)*t-1/(dx)*x+(dt+dx)/(dt*dx)
timestep(_);
```

```
↦ [1]:
↦    _[1,1]=t
↦    _[1,2]=0
↦    _[2,1]=0
↦    _[2,2]=t
↦ [2]:
↦    _[1,1]=(-dt)/(dx)*x+(dt+dx)/(dx)
↦    _[1,2]=0
↦    _[2,1]=0
↦    _[2,2]=(-dt)/(dx)*x+(dt+dx)/(dx)
```

D.15.4.17 fouriersystem

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: fouriersystem(M,A); M a list of matrices, A a list of approximations;

Return: type list; each entry is some matrix obtained by performing the substitution of the single approximations into the system of pde's, partitioning the equation into the several timesteps and fouriertransforming these parts

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
matrix M[2][2]=0,-a,-a,0;
list Mat=unitmat(2),M;
list Appr=forward(Ut,U,t),trapezoid(Ux,U,x);
def s=fouriersystem(Mat,Appr);s;
↦ [1]:
↦    [1]:
↦       [1]:
↦          _[1,1]=(I*T*Sx+T*Cx)
↦          _[1,2]=0
↦          _[2,1]=0
↦          _[2,2]=(I*T*Sx+T*Cx)
↦       [2]:
↦          _[1,1]=(I*Sx+Cx)
↦          _[1,2]=(2*I*Sx*a*dt+2*Cx*a*dt-2*a*dt)/(dx)
↦          _[2,1]=(2*I*Sx*a*dt+2*Cx*a*dt-2*a*dt)/(dx)
↦          _[2,2]=(I*Sx+Cx)
↦    [2]:
↦       [1]:
↦          _[1]=(I*T*Sx+T*Cx)
↦       [2]:
↦          _[1]=(-2*I*Sx*a*dt+I*Sx*dx-2*Cx*a*dt+Cx*dx+2*a*dt)/(dx)
↦          _[2]=(2*I*Sx*a*dt+I*Sx*dx+2*Cx*a*dt+Cx*dx-2*a*dt)/(dx)
↦ [2]:
↦    [1]:
↦       [1]:
↦          (T^2)
↦    [2]:
```

```
↦         [1]:
↦            (-8*Cx*a^2*dt^2+4*Cx*a*dt*dx+8*a^2*dt^2-4*a*dt*dx+dx^2)/(dx^2)
↦         [2]:
↦            (-8*Cx*a^2*dt^2-4*Cx*a*dt*dx+8*a^2*dt^2+4*a*dt*dx+dx^2)/(dx^2)
```

D.15.4.18 PartitionVar

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: PartitionVar(f); f a poly in the basering;

Return: type poly; gives back a list L=f1,f2 obtained by the partition of f into two parts f1,f2 with deg_var_n(f1) >0 deg_var_n(f2)=0

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);////does not show the ring, since there is no output
basering;///does show the ring
↦ // coefficients: QQ(I, T, Px, Py, Cx, Cy, Sx, Sy, a, b, dt, dx, dy)
↦ // number of vars : 8
↦ //        block   1 : ordering c
↦ //        block   2 : ordering lp
↦ //                  : names    i t x y cx cy sx sy
↦ // quotient ring from ideal
↦ _[1]=cy^2+sy^2-1
↦ _[2]=cx^2+sx^2-1
↦ _[3]=i^2+1
poly f=t**3*cx**2-cy**2*dt+i**3*sx;
PartitionVar(f,1); ////i is the first variable
↦ [1]:
↦    i^3*sx
↦ [2]:
↦    t^3*cx^2+(-dt)*cy^2
```

D.15.4.19 ComplexValue

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: ComplexValue(f); f a poly in the basering;

Return: type poly; gives back the formal complex-value of f, where var(1) is redarded as the imaginary unit. Does only make sense, if the proc <setinitials> is executed before -> nvars <= npars

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);////does not show the ring, as there is  no output
basering;///does show the ring
↦ // coefficients: QQ(I, T, Px, Py, Cx, Cy, Sx, Sy, a, b, dt, dx, dy)
↦ // number of vars : 8
↦ //        block   1 : ordering c
↦ //        block   2 : ordering lp
↦ //                  : names    i t x y cx cy sx sy
↦ // quotient ring from ideal
↦ _[1]=cy^2+sy^2-1
↦ _[2]=cx^2+sx^2-1
↦ _[3]=i^2+1
poly f=t**3*cx**2-cy**2*dt+i**3*sx;
f;
↦ i^3*sx+t^3*cx^2+(-dt)*cy^2
ComplexValue(f);
↦ t^6*sx^4-2*t^6*sx^2+t^6+(-2*dt)*t^3*sx^2*sy^2+(2*dt)*t^3*sx^2+(2*dt)*t^3*\
   sy^2+(-2*dt)*t^3+sx^2+(dt^2)*sy^4+(-2*dt^2)*sy^2+(dt^2)
```

D.15.4.20 VarToPar

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: VarToPar(f); f a poly in the basering;

Return: type poly; gives back the poly obtained by substituting var(i) by par(i), for all variables. Does only make sense, if the proc <setinitials> is executed before -> nvars <= npars;

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);////does not show the ring, as there is  no output
basering;///does show the ring
↦ // coefficients: QQ(I, T, Px, Py, Cx, Cy, Sx, Sy, a, b, dt, dx, dy)
↦ // number of vars : 8
↦ //        block   1 : ordering c
↦ //        block   2 : ordering lp
↦ //                  : names    i t x y cx cy sx sy
↦ // quotient ring from ideal
↦ _[1]=cy^2+sy^2-1
↦ _[2]=cx^2+sx^2-1
↦ _[3]=i^2+1
poly f=t**3*cx**2-cy**2*dt+i**3*sx;
f;
↦ i^3*sx+t^3*cx^2+(-dt)*cy^2
VarToPar(f);
↦ (I^3*Sx+T^3*Cx^2-Cy^2*dt)
```

D.15.4.21 ParToVar

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: ParToVar(f); f a poly in the basering;

Return: type poly; gives back the poly obtained by substituting par(i) by var(i), for the first nvars(basering parameters. Does only make sense, if setinitials is executed before -> nvars <= npars. Is the opposite action to VarToPar, see example ParToVar;

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);////does not show the ring, as there is  no output
basering;///does show the ring
↦ // coefficients: QQ(I, T, Px, Py, Cx, Cy, Sx, Sy, a, b, dt, dx, dy)
↦ // number of vars : 8
↦ //        block   1 : ordering c
↦ //        block   2 : ordering lp
↦ //                  : names    i t x y cx cy sx sy
↦ // quotient ring from ideal
↦ _[1]=cy^2+sy^2-1
↦ _[2]=cx^2+sx^2-1
↦ _[3]=i^2+1
poly f=t**3*cx**2-cy**2*dt+i**3*sx/dt*dx;
f;
↦ (dx)/(dt)*i^3*sx+t^3*cx^2+(-dt)*cy^2
def g=VarToPar(f);
g;
↦ (I^3*Sx*dx+T^3*Cx^2*dt-Cy^2*dt^2)/(dt)
def h=ParToVar(g);
h==f;
↦ 1
```

D.15.4.22 qepcad

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: qepcad(f); f a poly in the basering;

Return: type list; gives back some constraints that are equivalent to f<1 (computed by QEPCAD);

Example:

```
LIB "finitediff.lib";
list D="Ux","Ut","U";
list P="a";
list V="t","x";
setinitials(V,D,P);
def s1=scheme(u(Ut)+a*u(Ux),laxfrX(Ux,U,x),laxfrT(Ut,U,x));
s1;
↦ (-a*dt+dx)/(2*dx)*x^2+(a*dt+dx)/(2*dx)
def s2=errormap(s1);
s2;
↦ (-a*dt+dx)/(dx)*i*cx*sx+(a*dt-dx)/(dx)*sx^2+1
def s3=ComplexValue(s2);s3;
↦ (a^2*dt^2-dx^2)/(dx^2)*sx^2+1
qepcad(s3);
↦ 0
↦ 0
↦ 0
↦ sh: qepcad: Kommando nicht gefunden.
↦ sh: qepcadfilter.pl: Kommando nicht gefunden.
↦ 32512
↦
↦ Try manually
```

D.15.4.23 qepcadsystem

Procedure from library `finitediff.lib` (see Section D.15.4 [finitediff_lib], page 2329).

Usage: qepcadsytem(f); l a list;

Return: list

Purpose: gives back some constraints that are equivalent to the eigenvalues of the matrices in the list l being < 1 (computed by QEPCAD)

Example:

```
LIB "finitediff.lib";
list D="Ut","Ux","Uy","U";
list V="t","x","y";
list P="a","b";
setinitials(V,D,P);
matrix M[2][2]=0,-a,-a,0;
list Mat=unitmat(2),M;
list Appr=forward(Ut,U,t),forward(Ux,U,x);
//matrixsystem(Mat,Appr);
//timestep(_);
fouriersystem(Mat,Appr);
↦ [1]:
↦    [1]:
↦       [1]:
↦          _[1,1]=(T)
↦          _[1,2]=0
↦          _[2,1]=0
↦          _[2,2]=(T)
↦       [2]:
↦          _[1,1]=1
↦          _[1,2]=(I*Sx*a*dt+Cx*a*dt-a*dt)/(dx)
↦          _[2,1]=(I*Sx*a*dt+Cx*a*dt-a*dt)/(dx)
↦          _[2,2]=1
↦    [2]:
↦       [1]:
↦          _[1]=(T)
↦       [2]:
↦          _[1]=(-I*Sx*a*dt-Cx*a*dt+a*dt+dx)/(dx)
↦          _[2]=(I*Sx*a*dt+Cx*a*dt-a*dt+dx)/(dx)
↦ [2]:
↦    [1]:
↦       [1]:
↦          (T^2)
↦    [2]:
↦       [1]:
↦          (-2*Cx*a^2*dt^2-2*Cx*a*dt*dx+2*a^2*dt^2+2*a*dt*dx+dx^2)/(dx^2)
↦       [2]:
↦          (-2*Cx*a^2*dt^2+2*Cx*a*dt*dx+2*a^2*dt^2-2*a*dt*dx+dx^2)/(dx^2)
```

```
qepcadsystem(_[2]);
↦ 0
↦ 0
↦ 0
↦ sh: qepcad: Kommando nicht gefunden.
↦ sh: qepcadfilter.pl: Kommando nicht gefunden.
↦ 32512
↦
↦    ? Try manually
↦    ? leaving finitediff.lib::qepcadsystem (0)
```

D.15.5 GND_lib

Library : GND.lib

Author : Adrian Popescu, popescu@mathematik.uni-kl.de

Overview :

A method to compute the General Neron Desingularization in the frame of one dimensional local domains

References:

[1] A. Popescu, D. Popescu, "A method to compute the General Neron Desingularization in the frame of one dimensional local domains", arxiv.org/abs/1508.05511

Procedures:

D.15.5.1 desingularization

Procedure from library `GND.lib` (see Section D.15.5 [GND_lib], page 2342).

Usage : Returns as output a General Neron Desingularization as in the Paper http://arxiv.org/abs/1508.05511

Example:

```
LIB "GND.lib";
//Example 1
ring All = 0,(a1,a2,a3,x1,x2,x3,Y1,Y2,Y3),dp;
int nra = 3;
int nrx = 3;
int nry = 3;
ideal xid = x2^3-x3^2,x1^3-x3^2;
ideal yid = Y1^3-Y2^3;
ideal aid = a3^2-a1*a2;
poly y;
int i;
for(i=0;i<=30;i++)
{
y = y + a1*x3^i/factorial(i);
}
for(i=31;i<=50;i++)
{
y = y + a2*x3^i/factorial(i);
}
```

```
ideal f = a3*x1,a3*x2,y;
desingularization(All, nra,nrx,nry,xid,yid,aid,f);
↦ h =
↦ h[1]=Y1+(x1^3*x3^6)*T3+(-a3*x1)
↦ h[2]=Y2+(-x1^2*x3^6)*T1+(x1^2*x2*x3^6)*T3+(-a3*x2)
↦ h[3]=Y3+(x1^3*x3^6)*T2+(-a1*x3^7-7*a1*x3^6-42*a1*x3^5-210*a1*x3^4-840*a1*\
   x3^3-2520*a1*x3^2-5040*a1*x3-5040*a1)/5040
↦ h[4]=Y4+(-x1^2*x3^4)*T4+1
// With debug output
desingularization(All, nra,nrx,nry,xid,yid,aid,f,"debug");
↦ Computing the kernel:
↦ ker[1]=x2*Y1-x1*Y2
↦ ker[2]=Y1^3-Y2^3
↦ ker[3]=x1*Y1^2-x2*Y2^2
↦ ker[4]=x1^2*Y1-x2^2*Y2
↦ ker[5]=x2^3-x3^2
↦ ker[6]=x1^3-x3^2
↦ ker[7]=x1*x2^2*Y2-x3^2*Y1
↦ ker[8]=x1^2*x2*Y2^2-x3^2*Y1^2
↦ This is Plist:
↦ [1]:
↦    -x1*x3^2
↦ [2]:
↦    x3^2
↦ [3]:
↦    -x1
↦ [4]:
↦    _[1]=x2*Y1-x1*Y2
↦ [5]:
↦    [1]:
↦       1
↦ The minor comes from these vars:
↦ [1]:
↦    1
↦ [2]:
↦    2
↦ P’ =  -x1*x3^2
↦ v(P’):
↦ -x1*x3^2
↦ d’ = x1*x3^2
↦ z = -1
↦ P is constant (no Y), so d = d’ = P = P’
↦ P = P’ =  -x1*x3^2
↦ d =  -x1*x3^2
↦ vidjet:
↦ vidjet[1]=x1
↦ vidjet[2]=x2
↦ vidjet[3]=x3
↦ vidjet[4]=(a3)*x1
↦ vidjet[5]=(a3)*x2
↦ vidjet[6]=(a1)/5040*x3^7+(a1)/720*x3^6+(a1)/120*x3^5+(a1)/24*x3^4+(a1)/6*\
   x3^3+(a1)/2*x3^2+(a1)*x3+(a1)
↦ vidjet[7]=-1
↦ Py = -x1*x3^2
↦ This is C:
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    a1 a3 x1 x2 x3
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=x2^3-x3^2
↦ _[2]=x1^3-x3^2
↦ _[3]=x3^8
↦ This is D:
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    a1 a3 x1 x2 x3
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=x2^3-x3^2
↦ _[2]=x1^3-x3^2
↦ This is the minor bordered matrix (H)
↦ (x2),(-x1),0,0,
↦ 0,   0,    1,0,
↦ 1,   0,    0,0,
↦ 0,   0,    0,(-x1*x3^2)
↦ This is G:
↦ 0,              0,                (x1^2*x3^4)*Y4^2, 0,
↦ (-x1*x3^4)*Y4^2,0,                (x1*x2*x3^4)*Y4^2,0,
↦ 0,              (x1^2*x3^4)*Y4^2,0,                 0,
↦ 0,              0,                0,                (-x1*x3^2)*Y4^2
↦ G[1,1]=0
↦ G[1,2]=0
↦ G[1,3]=(x1^2*x3^4)*Y4^2
↦ G[1,4]=0
↦ G[2,1]=(-x1*x3^4)*Y4^2
↦ G[2,2]=0
↦ G[2,3]=(x1*x2*x3^4)*Y4^2
↦ G[2,4]=0
↦ G[3,1]=0
↦ G[3,2]=(x1^2*x3^4)*Y4^2
↦ G[3,3]=0
↦ G[3,4]=0
↦ G[4,1]=0
↦ G[4,2]=0
↦ G[4,3]=0
↦ G[4,4]=(-x1*x3^2)*Y4^2
↦ s = 1
↦ This is cc
↦ cc[1]=0
↦ cc[2]=0
↦ h =
↦ h[1]=Y1+(x1^3*x3^6)*T3+(-a3*x1)
↦ h[2]=Y2+(-x1^2*x3^6)*T1+(x1^2*x2*x3^6)*T3+(-a3*x2)
↦ h[3]=Y3+(x1^3*x3^6)*T2+(-a1*x3^7-7*a1*x3^6-42*a1*x3^5-210*a1*x3^4-840*a1*\
   x3^3-2520*a1*x3^2-5040*a1*x3-5040*a1)/5040
↦ h[4]=Y4+(-x1^2*x3^4)*T4+1
↦ m = 1
↦ s^m = 1
↦ QT =
↦ QT[1]=0
↦ QT[2]=0
↦ f =
↦ f[1]=(x2)*Y1+(-x1)*Y2
↦ f[2]=(-x1*x3^2)*Y4+(x1*x3^2)
↦ g =
↦ g[1]=T1
↦ g[2]=T2
kill All,nra,nrx,nry,i;
//Example 4
ring All = 0,(a1,a2,a3,a4,x,Y1,Y2,Y3),dp;
int nra = 4;
int nrx = 1;
int nry = 3;
ideal xid = 0;
ideal yid = Y1^3-Y2^3;
ideal aid = a3^2-a1*a2,a4^2+a4+1;
poly y;
int i;
for(i=0;i<=30;i++)
{
y = y + a1*x3^i/factorial(i);
}
for(i=31;i<=50;i++)
{
y = y + a2*x3^i/factorial(i);
}
ideal f = a3*x,a3*a4*x,y;
desingularization(All, nra,nrx,nry,xid,yid,aid,f);
↦ h =
↦ _[1]=Y1+(-4*a3^2*a4^2*x^6-4*a3^2*a4*x^6-a3^2*x^6-8*a3*a4*x^6-4*a3*x^6-4*x\
   ^6)/(4*a4^2+4*a4+1)*T3+(-a3*x)
↦ _[2]=Y2+(-2*a3*a4*x^5-a3*x^5-2*x^5)/(2*a4+1)*T1+(8*a3^2*a4^2*x^6+8*a3^2*a\
   4*x^6+2*a3^2*x^6+10*a3*a4*x^6+5*a3*x^6+2*x^6)/(4*a4^2+4*a4+1)*T3+(-x)/(2*\
   a4+1)
↦ _[3]=Y3+(-4*a3^2*a4^2*x^6-4*a3^2*a4*x^6-a3^2*x^6-8*a3*a4*x^6-4*a3*x^6-4*x\
   ^6)/(4*a4^2+4*a4+1)*T2+(-a1*x^3-a1)
↦ _[4]=Y4+(2*x^5)*T1+(-3*a3*x^6)*T3+(-2*a3*a4*x^5-a3*x^5-2*x^5)/(2*a4+1)*T4\
   +(-x)
```

D.15.6 gradedModules_lib

Library: gradedModules.lib

Purpose: Operations with graded modules/matrices/resolutions

Authors: Oleksandr Motsak <U@D>, where U=motsak, D=mathematik.uni-kl.de
Hanieh Keneshlou <hkeneshlou@yahoo.com>

Overview: The library contains several procedures for constructing and manipulating graded modules/matrices/resolutions. Basics about graded objects can be found in [DL].
Throughout this library graded objects are graded maps, that is, matrices with polynomials, together with grading weights for source and destination. Graded modules are implicitly given as coker of a graded map. Note that in special cases we may also consider submodules in S^r generated by columns of a graded polynomial matrix (or a graded map).

Note: set assumeLevel to positive integer value in order to auto-check all assumptions. We denote the current basering by S.

References:

[DL] Decker, W., Lossen, Ch.: Computing in Algebraic Geometry, Springer, 2006

Procedures:

D.15.6.1 grobj

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grobj(M, w[, d]), matrix/ideal/module M, intvec w, d

Return: graded object with matrix presentation M, row weighting w [and total graded degrees d of columns]

Purpose: create a valid graded object with a given matrix presentation, weighting [and total graded degrees (in case of zero columns)]

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
def A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
  ees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   - |..1
↦    0 :  1   1 |..2
↦    0 :  -   1 |..3
↦    1 :  -   - |..4
↦       === ===
↦         1   1
def F = grobj( module([x,y,0]), intvec(1,1,5) );
grview(F);
↦ Graded homomorphism: r(-1)^2 + r(-5) <- r(-2), given by a matrix, with de\
  grees:
↦       ..1 ....
↦       --- +...
↦    1 :  1 |..1
↦    1 :  1 |..2
↦    5 :  - |..3
↦       ===
↦         2
int d = 666; // zero can have any degree...
def Z = grobj( module([x,0], [0,0,0], [0, y]), intvec(1,2,3), intvec(2, d, 3) );
```

```
grview(Z);
↦ Graded homomorphism: r(-1) + r(-2) + r(-3) <- r(-2) + r(-666) + r(-3), gi\
  ven by a square matrix, with degrees:
↦       ...1 ...2 ...3 .....
↦       ---- ---- ---- +....
↦    1 :   1    -    - |...1
↦    2 :   -    -    1 |...2
↦    3 :   -    -    - |...3
↦       ==== ==== ====
↦          2  666    3
print(Z);
↦ x,0,0,
↦ 0,0,y,
↦ 0,0,0
attrib(Z);
↦ attr:degHomog, type intvec
↦ attr:isHomog, type intvec
grrange(Z); // module weights
↦ 1,2,3
attrib(Z, "degHomog"); // total degrees
↦ 2,666,3
// Zero object:
matrix z[3][0];  grview( grobj( z, intvec(1,2,3) ) );
↦ Graded homomorphism: r(-1) + r(-2) + r(-3) <- 0, given by zero (3 x 0) ma\
  trix.
grview( grobj( freemodule(0), intvec(1,2,3) ) );
↦ Graded homomorphism: r(-1) + r(-2) + r(-3) <- 0, given by zero (3 x 0) ma\
  trix.
matrix z1[0][3]; grview( grobj( z1, 0:0, intvec(1,2,3) ) );
↦ Graded homomorphism: 0 <- r(-1) + r(-2) + r(-3), given by zero (0 x 3) ma\
  trix.
grview( grobj( freemodule(0), 0:0, intvec(1,2,3) ) );
↦ Graded homomorphism: 0 <- r(-1) + r(-2) + r(-3), given by zero (0 x 3) ma\
  trix.
matrix z0[0][0]; grview( grobj( z0, 0:0 ) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
grview( grobj( freemodule(0), 0:0 ) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
```

D.15.6.2 grtest

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grtest(M[,b]), anything M, optionally int b

Return: 1 if M is a valid graded object, 0 otherwise

Purpose: validate a graded object. Print an invalid object message if b is not given

Note: M should be an ideal or module or matrix, with weighting attribute 'isHomog' and optionally total graded degrees attribute 'degHomog'. Attributes should be compatible with the presentation matrix.

Example:

```
LIB "gradedModules.lib";
```

```
ring r=32003,(x,y,z),dp;
// the following calls will fail due to tests in grtest:
grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,0) ); // enough row weigh
↦  _[1]=x*gen(2)+x*gen(1)+y*gen(1)
↦  _[2]=x*gen(2)+y*gen(3)+y*gen(2)
// grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0) ); // not enough row we
// grobj( module([x,0], [0,0,0], [0, y]), intvec(1,2,3) ); // zero column needs (othe
grobj( module([x,0], [0,0,0], [0, y]), intvec(1,2,3), intvec(2, 10, 3) ); // compatib
↦  _[1]=x*gen(1)
↦  _[2]=0
↦  _[3]=y*gen(2)
// grobj( module([x,0], [0,0,0], [0, y]), intvec(1,2,3), intvec(2-1, 10, 3+1) ); // i
```

D.15.6.3 grdeg

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grdeg(M), graded object M

Return: intvec of degrees

Purpose: graded degrees of columns (generators) of M, describing the source of M

Assume: M must be a graded object (matrix/module/ideal/mapping)

Note: if M has zero cols it should have attrib(M,'degHomog') set.

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦  Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
   ees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :   1   - |..1
↦    0 :   1   1 |..2
↦    0 :   -   1 |..3
↦    1 :   -   - |..4
↦       === ===
↦         1   1
module B = grobj( module([0,x,y]), intvec(15,1,1) );
grview(B);
↦  Graded homomorphism: r(-15) + r(-1)^2 <- r(-2), given by a matrix, with d\
   egrees:
↦       ..1 ....
↦       --- +...
↦   15 :   - |..1
↦    1 :   1 |..2
↦    1 :   1 |..3
↦       ===
↦         2
module D = grsum(
grsum(grpower(A,2), grtwist(1,1)),
grsum(grtwist(1,2), grpower(B,2))
);
grview(D);
↦ Graded homomorphism:
↦ r^3 + r(-1) + r^3 + r(-1) + r(1) + r(2) + r(-15) + r(-1)^2 + r(-15) + r(-\
  1)^2 <-
↦ r(-1)^4 + r(-2)^2, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   0 :  1   -   -   -   -   - |..1
↦   0 :  1   1   -   -   -   - |..2
↦   0 :  -   1   -   -   -   - |..3
↦   1 :  -   -   -   -   -   - |..4
↦   0 :  -   -   1   -   -   - |..5
↦   0 :  -   -   1   1   -   - |..6
↦   0 :  -   -   -   1   -   - |..7
↦   1 :  -   -   -   -   -   - |..8
↦  -1 :  -   -   -   -   -   - |..9
↦  -2 :  -   -   -   -   -   - |.10
↦  15 :  -   -   -   -   -   - |.11
↦   1 :  -   -   -   -   1   - |.12
↦   1 :  -   -   -   -   1   - |.13
↦  15 :  -   -   -   -   -   - |.14
↦   1 :  -   -   -   -   -   1 |.15
↦   1 :  -   -   -   -   -   1 |.16
↦      === === === === === ===
↦        1   1   1   1   2   2
grdeg(D);
↦ 1,1,1,1,2,2
def D10 = grshift(D, 10);
grview(D10);
↦ Graded homomorphism:
↦ r(10)^3 + r(9) + r(10)^3 + r(9) + r(11) + r(12) + r(-5) + r(9)^2 + r(-5) \
  + r(9)^2 <-
↦ r(9)^4 + r(8)^2, given by a matrix, with degrees:
↦       ...1 ...2 ...3 ...4 ...5 ...6 .....
↦       ---- ---- ---- ---- ---- ---- +....
↦  -10 :   1    -    -    -    -    - |...1
↦  -10 :   1    1    -    -    -    - |...2
↦  -10 :   -    1    -    -    -    - |...3
↦   -9 :   -    -    -    -    -    - |...4
↦  -10 :   -    -    1    -    -    - |...5
↦  -10 :   -    -    1    1    -    - |...6
↦  -10 :   -    -    -    1    -    - |...7
↦   -9 :   -    -    -    -    -    - |...8
↦  -11 :   -    -    -    -    -    - |...9
↦  -12 :   -    -    -    -    -    - |..10
↦    5 :   -    -    -    -    -    - |..11
↦   -9 :   -    -    -    -    1    - |..12
↦   -9 :   -    -    -    -    1    - |..13
↦    5 :   -    -    -    -    -    - |..14
↦   -9 :   -    -    -    -    -    1 |..15
↦   -9 :   -    -    -    -    -    1 |..16
↦       ==== ==== ==== ==== ==== ====
↦           -9    -9    -9    -9    -8    -8
grdeg(D10);
↦ -9,-9,-9,-9,-8,-8
```

D.15.6.4 grview

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grview(M), graded object M

Return: nothing

Purpose: print the degree/grading data about the GRADED matrix/module/ideal/mapping object M

Assume: M must be graded

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
   ees:
↦        ..1 ..2 ....
↦        --- --- +...
↦    0 :   1   - |..1
↦    0 :   1   1 |..2
↦    0 :   -   1 |..3
↦    1 :   -   - |..4
↦        === ===
↦          1   1
module B = grobj( module([0,x,y]), intvec(15,1,1) );
grview(B);
↦ Graded homomorphism: r(-15) + r(-1)^2 <- r(-2), given by a matrix, with d\
   egrees:
↦        ..1 ....
↦        --- +...
↦   15 :   - |..1
↦    1 :   1 |..2
↦    1 :   1 |..3
↦        ===
↦          2
module D = grsum( grsum(grpower(A,2), grtwist(1,1)), grsum(grtwist(1,2), grpower(B,2
grview(D);
↦ Graded homomorphism:
↦ r^3 + r(-1) + r^3 + r(-1) + r(1) + r(2) + r(-15) + r(-1)^2 + r(-15) + r(-\
   1)^2 <-
↦ r(-1)^4 + r(-2)^2, given by a matrix, with degrees:
↦        ..1 ..2 ..3 ..4 ..5 ..6 ....
↦        --- --- --- --- --- --- +...
↦    0 :   1   -   -   -   -   - |..1
↦    0 :   1   1   -   -   -   - |..2
↦    0 :   -   1   -   -   -   - |..3
↦    1 :   -   -   -   -   -   - |..4
↦   0 :  -   -   1   -   -   - |..5
↦   0 :  -   -   1   1   -   - |..6
↦   0 :  -   -   -   1   -   - |..7
↦   1 :  -   -   -   -   -   - |..8
↦  -1 :  -   -   -   -   -   - |..9
↦  -2 :  -   -   -   -   -   - |.10
↦  15 :  -   -   -   -   -   - |.11
↦   1 :  -   -   -   -   1   - |.12
↦   1 :  -   -   -   -   1   - |.13
↦  15 :  -   -   -   -   -   - |.14
↦   1 :  -   -   -   -   -   1 |.15
↦   1 :  -   -   -   -   -   1 |.16
↦      === === === === === ===
↦        1   1   1   1   2   2
ring R = 0,(w,x,y,z), dp; def I = grobj( ideal(y2-xz, xy-wz, x2z-wyz), intvec(0) );
list res1 = grres(I, 0); // non-minimal
grview(res1);
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-2)^2 + R(-3) <-- d_2 --
↦ R(-3) + R(-4) <-- d_3 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-2)^2 + R(-3), given by a matrix, with degree\
  s:
↦     .1 .2 .3 ...
↦     -- -- -- +..
↦  0 : 2  2  3 |.1
↦     == == ==
↦      2  2  3
↦ d_2 :
↦ Graded homomorphism: R(-2)^2 + R(-3) <- R(-3) + R(-4), given by a matrix,\
   with degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  2 : 1  2 |.1
↦  2 : 1  2 |.2
↦  3 : 0  1 |.3
↦     == ==
↦      3  4
↦ d_3 :
↦ Graded homomorphism: R(-3) + R(-4) <- 0, given by zero (2 x 0) matrix.
print(betti(res1,0), "betti");
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     2     1
↦     2:     -     1     1
↦ ------------------------
↦ total:     1     3     2
↦
list res2 = grres(grshift(I, -10), 0, 1); //  minimal!
grview(res2);
```

```
↦ Graded resolution:
↦ R(-10) <-- d_1 --
↦ R(-12)^2 <-- d_2 --
↦ R(-14) <-- d_3 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(-10) <- R(-12)^2, given by a matrix, with degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦  10 :   2   2 |..1
↦      === ===
↦       12  12
↦ d_2 :
↦ Graded homomorphism: R(-12)^2 <- R(-14), given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  12 :   2 |..1
↦  12 :   2 |..2
↦      ===
↦       14
↦ d_3 :
↦ Graded homomorphism: R(-14) <- 0, given by zero (1 x 0) matrix.
print(betti(res2,0), "betti");
↦            0     1     2
↦ ------------------------
↦    10:     1     -     -
↦    11:     -     2     -
↦    12:     -     -     1
↦ ------------------------
↦ total:     1     2     1
↦
```

D.15.6.5 grshift

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grshift(A, d), graded objects A, int d

Return: shifted graded object

Purpose: shift the grading on A by d: A(i) -> A(i+d)

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
  ees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   0 :   1   - |..1
↦   0 :   1   1 |..2
↦   0 :   -   1 |..3
↦   1 :   -   - |..4
↦       === ===
↦         1   1
module S = grshift( A,  6);
grview(S);
↦ Graded homomorphism: r(6)^3 + r(5) <- r(5)^2, given by a matrix, with deg\
   rees:
↦       ..1 ..2 ....
↦       --- --- +...
↦   -6 :  1   - |..1
↦   -6 :  1   1 |..2
↦   -6 :  -   1 |..3
↦   -5 :  -   - |..4
↦       === ===
↦        -5  -5
grview( grshift( grzero(), 100 ) ); // does nothing...
↦ !! Warning: shifting '0 <- 0' leaves it as it unchanged!
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
```

D.15.6.6 grzero

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grzero()

Return: graded object representing S(0)^1

Purpose: compute presentation of S(0)^1

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
grview( grobj(freemodule(0), intvec(0:0), intvec(0:0)) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
grview( grobj(freemodule(0), intvec(0:0)) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
grview( grzero() );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
//  def M = grpower( grshift( grzero(), 3), 2 ); grview(M);
```

D.15.6.7 grtwist

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grtwist(a,d), int a, d

Return: graded object representing S(d)^a

Purpose: compute presentation of S(d)^a

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
//  grview(grpower( grshift(grzero(), 10), 5 ) );
grview( grtwist (5, 10) );
↦ Graded homomorphism: r(10)^5 <- 0, given by zero (5 x 0) matrix.
```

D.15.6.8 grtwists

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grtwists(v), intvec v

Return: graded object representing S(v[1]) + ... + S(v[size(v)])

Purpose: compute presentation of S(v[1]) + ... + S(v[size(v)])

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
grview( grtwists ( intvec(-4, 1, 6 )) );
↦ Graded homomorphism: r(-4) + r(1) + r(6) <- 0, given by zero (3 x 0) matr\
   ix.
grview( grtwists ( intvec(0:0) ) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
```

D.15.6.9 grsum

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grsum(A, B), graded objects A and B

Return: graded direct sum of input objects

Purpose: compute the graded direct sum of A and B

Example:

```
LIB "gradedModules.lib";
//  if( defined(assumeLevel) ){ int assumeLevel0 = assumeLevel; } else { int assumeLe
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
   ees:
↦        ..1 ..2 ....
↦        --- --- +...
↦    0 :   1   - |..1
↦    0 :   1   1 |..2
↦    0 :   -   1 |..3
↦    1 :   -   - |..4
↦        === ===
↦          1   1
module B = grobj( module([0,x,y]), intvec(15,1,1) );
grview(B);
↦ Graded homomorphism: r(-15) + r(-1)^2 <- r(-2), given by a matrix, with d\
   egrees:
↦        ..1 ....
↦        --- +...
↦   15 :   - |..1
↦    1 :   1 |..2
↦    1 :   1 |..3
↦        ===
↦          2
module C = grsum(A,B);
```

```
print(C);
↦ x+y,0,  0,
↦ x,  x+y,0,
↦ 0,  y,  0,
↦ 0,  0,  0,
↦ 0,  0,  0,
↦ 0,  0,  x,
↦ 0,  0,  y
homog(C);
↦ 1
grview(C);
↦ Graded homomorphism: r^3 + r(-1) + r(-15) + r(-1)^2 <- r(-1)^2 + r(-2), g\
  iven by a matrix, with degrees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :  1   -   - |..1
↦   0 :  1   1   - |..2
↦   0 :  -   1   - |..3
↦   1 :  -   -   - |..4
↦  15 :  -   -   - |..5
↦   1 :  -   -   1 |..6
↦   1 :  -   -   1 |..7
↦      === === ===
↦        1   1   2
module D = grsum(
grsum(grpower(A,2), grtwist(1,1)),
grsum(grtwist(1,2), grpower(B,2))
);
print(D);
↦ x+y,0,  0,  0,  0,0,
↦ x,  x+y,0,  0,  0,0,
↦ 0,  y,  0,  0,  0,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  x+y,0,  0,0,
↦ 0,  0,  x,  x+y,0,0,
↦ 0,  0,  0,  y,  0,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  0,  0,  x,0,
↦ 0,  0,  0,  0,  y,0,
↦ 0,  0,  0,  0,  0,0,
↦ 0,  0,  0,  0,  0,x,
↦ 0,  0,  0,  0,  0,y
homog(D);
↦ 1
grview(D);
↦ Graded homomorphism:
↦ r^3 + r(-1) + r^3 + r(-1) + r(1) + r(2) + r(-15) + r(-1)^2 + r(-15) + r(-\
  1)^2 <-
↦ r(-1)^4 + r(-2)^2, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   0 :  1   -   -   -   -   - |..1
↦   0 :  1   1   -   -   -   - |..2
↦   0 :  -   1   -   -   -   - |..3
↦   1 :  -   -   -   -   -   - |..4
↦   0 :  -   -   1   -   -   - |..5
↦   0 :  -   -   1   1   -   - |..6
↦   0 :  -   -   -   1   -   - |..7
↦   1 :  -   -   -   -   -   - |..8
↦  -1 :  -   -   -   -   -   - |..9
↦  -2 :  -   -   -   -   -   - |.10
↦  15 :  -   -   -   -   -   - |.11
↦   1 :  -   -   -   -   1   - |.12
↦   1 :  -   -   -   -   1   - |.13
↦  15 :  -   -   -   -   -   - |.14
↦   1 :  -   -   -   -   -   1 |.15
↦   1 :  -   -   -   -   -   1 |.16
↦      === === === === === ===
↦        1   1   1   1   2   2
module F = grobj( module([x,y,0]), intvec(1,1,5) );
grview(F);
↦ Graded homomorphism: r(-1)^2 + r(-5) <- r(-2), given by a matrix, with de\
  grees:
↦      ..1 ....
↦      --- +...
↦   1 :  1 |..1
↦   1 :  1 |..2
↦   5 :  - |..3
↦      ===
↦        2
module T = grsum( F, grsum( grtwist(1, 10), B ) );
grview(T);
↦ Graded homomorphism: r(-1)^2 + r(-5) + r(10) + r(-15) + r(-1)^2 <- r(-2)^\
  2
↦ , given by a matrix, with degrees:
↦       ...1 ...2 .....
↦       ---- ---- +....
↦    1 :   1    - |...1
↦    1 :   1    - |...2
↦    5 :   -    - |...3
↦  -10 :   -    - |...4
↦   15 :   -    - |...5
↦    1 :   -    1 |...6
↦    1 :   -    1 |...7
↦       ==== ====
↦          2    2
//  if( defined(assumeLevel0) ){ assumeLevel = assumeLevel0; } else { kill assumeLeve
```

D.15.6.10 grpower

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grpower(A, p), graded object A, int p > 0

Return: graded direct power A^p

Purpose: compute the graded direct power A^p

Note: the power p must be positive

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0], [0, x+y, y]), intvec(1,1,1) );
grview(A);
↦ Graded homomorphism: r(-1)^3 <- r(-2)^2, given by a matrix, with degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   1 :  1   - |..1
↦   1 :  1   1 |..2
↦   1 :  -   1 |..3
↦      === ===
↦        2   2
module B = grobj( module([x,y]), intvec(2,2) );
grview(B);
↦ Graded homomorphism: r(-2)^2 <- r(-3), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  2 : 1 |.1
↦  2 : 1 |.2
↦     ==
↦      3
module D = grsum( grpower(A,2), grpower(B,2) );
print(D);
↦ x+y,0,  0,  0,  0,0,
↦ x,  x+y,0,  0,  0,0,
↦ 0,  y,  0,  0,  0,0,
↦ 0,  0,  x+y,0,  0,0,
↦ 0,  0,  x,  x+y,0,0,
↦ 0,  0,  0,  y,  0,0,
↦ 0,  0,  0,  0,  x,0,
↦ 0,  0,  0,  0,  y,0,
↦ 0,  0,  0,  0,  0,x,
↦ 0,  0,  0,  0,  0,y
homog(D);
↦ 1
grview(D);
↦ Graded homomorphism: r(-1)^6 + r(-2)^4 <- r(-2)^4 + r(-3)^2, given by a m\
  atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   1 :  1   -   -   -   -   - |..1
↦   1 :  1   1   -   -   -   - |..2
↦   1 :  -   1   -   -   -   - |..3
↦   1 :  -   -   1   -   -   - |..4
↦   1 :  -   -   1   1   -   - |..5
↦   1 :  -   -   -   1   -   - |..6
↦   2 :  -   -   -   -   1   - |..7
↦    2 :  -   -   -   -   1   - |..8
↦    2 :  -   -   -   -   -   1 |..9
↦    2 :  -   -   -   -   -   1 |.10
↦       === === === === === ===
↦         2   2   2   2   3   3
```

D.15.6.11 grtranspose

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grtranspose(M), graded object M

Return: graded object

Purpose: graded transpose of M

Note: no reordering is performend by this procedure

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module M = grtwists( intvec(-2, 0, 4, 4) ); grview(M);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- 0, given by zero (4 x 0) matri\
   x.
module N = grsyz( grtranspose( M ) ); grview(N);
↦ Graded homomorphism: r(2) + r + r(-4)^2 <- r(2) + r + r(-4)^2, given by a\
    diagonal matrix, with degrees:
↦       ..1 ..2 ..3 ..4 ....
↦       --- --- --- --- +...
↦   -2 :  0   -   -   - |..1
↦    0 :  -   0   -   - |..2
↦    4 :  -   -   0   - |..3
↦    4 :  -   -   -   0 |..4
↦       === === === ===
↦        -2   0   4   4
module L = grtranspose(N); grview( L );
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- r(-2) + r + r(4)^2, given by a\
    diagonal matrix, with degrees:
↦       ..1 ..2 ..3 ..4 ....
↦       --- --- --- --- +...
↦    2 :  0   -   -   - |..1
↦    0 :  -   0   -   - |..2
↦   -4 :  -   -   0   - |..3
↦   -4 :  -   -   -   0 |..4
↦       === === === ===
↦         2   0  -4  -4
module K = grsyz( L ); grview(K);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- 0, given by zero (4 x 0) matri\
   x.
// Corner cases: 0 <- 0!
module Z = grzero(); grview(Z);
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
grview( grtranspose( Z ) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
// Corner cases: * <- 0
```

```
matrix M1[3][0];
module Z1 = grobj( M1, intvec(-1, 0, 1) ); grview(Z1);
↦ Graded homomorphism: r(1) + r + r(-1) <- 0, given by zero (3 x 0) matrix.
grview( grtranspose( Z1 ) );
↦ Graded homomorphism: 0 <- r(-1) + r + r(1), given by zero (0 x 3) matrix.
// Corner cases: 0 <- *
matrix M2[0][3];
module Z2 = grobj( M2, 0:0, intvec(-1, 0, 1) ); grview(Z2);
↦ Graded homomorphism: 0 <- r(1) + r + r(-1), given by zero (0 x 3) matrix.
grview( grtranspose( Z2 ) );
↦ Graded homomorphism: r(-1) + r + r(1) <- 0, given by zero (3 x 0) matrix.
```

D.15.6.12 grgens

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grgens(M), graded object M (map)

Return: graded object

Purpose: try compute graded generators of coker(M) and return them as columns of a graded map.

Note: presentation of resulting generated submodule may be different to M!

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module M = grtwists( intvec(-2, 0, 4, 4) ); grview(M);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- 0, given by zero (4 x 0) matri\
   x.
module N = grgens(M);
grview( N ); print(N); // fine == M
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- r(-2) + r + r(4)^2, given by a\
    diagonal matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦   2 :  0   -   -   - |..1
↦   0 :  -   0   -   - |..2
↦  -4 :  -   -   0   - |..3
↦  -4 :  -   -   -   0 |..4
↦      === === === ===
↦        2   0  -4  -4
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,0,1,0,
↦ 0,0,0,1
module A = grobj( module([x+y, x, 0, 3], [0, x+y, y, 2], [y, y, z, 1]), intvec(0,0,0
A = grgroebner(A); grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3 + r(-2) + r(-3), given by a m\
   atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   1   1   2   - |..1
↦   0 :  1   -   1   -   - |..2
↦   0 :  1   1   1   2   3 |..3
↦   1 :  0   0   0   1   2 |..4
↦      === === === === ===
↦        1   1   1   2   3
module B = grgens(A);
grview( B ); print(B); // Ups :( != A
↦ Graded homomorphism: r(2) <- r^3 + r(-1), given by a matrix, with degrees\
   :
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦  -2 :  2   2   2   3 |..1
↦      === === === ===
↦        0   0   0   1
↦ xy-3y2+xz+3yz,-xy+2y2+2xz+2yz,x2-xy-4y2,y3-x2z-2xyz-y2z
grview( grgens( grzero() ) );
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
```

D.15.6.13 grpres

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grpres(M), graded object M (submodule gens)

Return: graded module (via coker)

Purpose: compute graded presentation matrix of submodule generated by columns of M

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
def A = grgroebner( grobj( module([x+y, x, 0, 3], [0, x+y, y, 2], [y, y, z, 1]), intv
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3 + r(-2) + r(-3), given by a m\
   atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   1   1   2   - |..1
↦   0 :  1   -   1   -   - |..2
↦   0 :  1   1   1   2   3 |..3
↦   1 :  0   0   0   1   2 |..4
↦      === === === === ===
↦        1   1   1   2   3
"graded transpose: "; def B = grtranspose(A); grview( B ); print(B);
↦ graded transpose:
↦ Graded homomorphism: r(1)^3 + r(2) + r(3) <- r^3 + r(1), given by a matri\
   x, with degrees:
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦  -1 :  1   1   1   0 |..1
↦  -1 :  1   -   1   0 |..2
↦  -1 :  1   1   1   0 |..3
↦  -2 :  2   -   2   1 |..4
↦  -3 :  -   -   3   2 |..5
↦      === === === ===
↦        0   0   0  -1
↦ y,   y,z,              1,
↦ x+2y,0,-y+z,           2,
↦ -y,  x,y-z,            1,
↦ y2,  0,-xz,            -x+3y,
↦ 0,   0,y3-x2z-2xyz-y2z,-x2+xy+4y2
"... syzygy: "; def C = grsyz(B); grview(C);
↦ ... syzygy:
↦ Graded homomorphism: r^3 + r(1) <- r(-2), given by a matrix, with degrees\
   :
↦      ..1 ....
↦      --- +...
↦   0 :  2 |..1
↦   0 :  2 |..2
↦   0 :  2 |..3
↦  -1 :  3 |..4
↦      ===
↦        2
"... transposed: "; def D = grtranspose(C); grview( D ); print (D);
↦ ... transposed:
↦ Graded homomorphism: r(2) <- r^3 + r(-1), given by a matrix, with degrees\
   :
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦  -2 :  2   2   2   3 |..1
↦      === === === ===
↦        0   0   0   1
↦ xy-3y2+xz+3yz,-xy+2y2+2xz+2yz,x2-xy-4y2,y3-x2z-2xyz-y2z
"... and back to presentation: "; def E = grsyz( D ); grview(E); print(E);
↦ ... and back to presentation:
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3, given by a matrix, with degr\
   ees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :  1   1   1 |..1
↦   0 :  1   1   1 |..2
↦   0 :  1   1   1 |..3
↦   1 :  0   -   - |..4
↦      === === ===
↦        1   1   1
↦ y,x,   x-2y,
↦ y,-2y, x-3y,
↦ z,-y-z,-3z,
↦ 1,0,   0
def F = grgens( E ); grview(F); print(F);
↦ Graded homomorphism: r(2) <- r^3 + r(-1), given by a matrix, with degrees\
   :
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦  -2 :  2   2   2   3 |..1
↦      === === === ===
↦        0   0   0   1
↦ xy-3y2+xz+3yz,-xy+2y2+2xz+2yz,x2-xy-4y2,y3-x2z-2xyz-y2z
def G = grpres( F ); grview(G); print(G);
```

```
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3, given by a matrix, with degr\
  ees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :   1   1   1 |..1
↦   0 :   1   1   1 |..2
↦   0 :   1   1   1 |..3
↦   1 :   0   -   - |..4
↦      === === ===
↦        1   1   1
↦ y,x,   x-2y,
↦ y,-2y, x-3y,
↦ z,-y-z,-3z,
↦ 1,0,   0
def M = grtwists( intvec(-2, 0, 4, 4) ); grview(M);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- 0, given by zero (4 x 0) matri\
  x.
def N = grgens(M); grview( N ); print(N);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- r(-2) + r + r(4)^2, given by a\
   diagonal matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦   2 :   0   -   -   - |..1
↦   0 :   -   0   -   - |..2
↦  -4 :   -   -   0   - |..3
↦  -4 :   -   -   -   0 |..4
↦      === === === ===
↦        2   0  -4  -4
↦ 1,0,0,0,
↦ 0,1,0,0,
↦ 0,0,1,0,
↦ 0,0,0,1
def L = grpres( N ); grview( L ); print(L);
↦ Graded homomorphism: r(-2) + r + r(4)^2 <- 0, given by zero (4 x 0) matri\
  x.
↦ 4 x 0 zero matrix
```

D.15.6.14 grorder

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grorder(M), graded object or list M

Return: same as input

Purpose: reorder/transform graded object or chain complex M into block form

Assume: M must be a graded object or a list of graded objects

Example:

```
LIB "gradedModules.lib";
"Surface Name: 'rat.d10.g9.quart2' in P^4";
↦ Surface Name: 'rat.d10.g9.quart2' in P^4
int @p=31991; ring R = (@p),(x,y,z,u,v), dp;
ideal J = x3yu2-48/11x2y2u2-8356xy3u2+35/121y4u2+31/66x3zu2-54/83x2yzu2-61/18xy2zu2+
def I = grobj( groebner(J), intvec(0) ); // ASSUME: no zero entries in J!
ASSUME(0, grtest(I));
"Input degrees: "; grview(I);
↦ Input degrees:
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^6 + R(-6), given by a matrix, w\
  ith degrees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 .9 ...
↦     -- -- -- -- -- -- -- -- -- +..
↦  0 : 4  4  5  5  5  5  5  5  6 |.1
↦     == == == == == == == == ==
↦      4  4  5  5  5  5  5  5  6
def RR = grres(I, 0, 1);
list L = RR;
" = Non-minimal betti numbers: ";  print(betti(L, 0), "betti");
↦  = Non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     2     -     -     -
↦     4:     -     5     9     3     -
↦     5:     -     1     3     3     1
↦ ------------------------------------
↦ total:     1     8    12     6     1
↦ 
"Graded reordered structure of 'res(Input,0)': ";  grview(grorder(L));
↦ Graded reordered structure of 'res(Input,0)':
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-4)^2 + R(-5)^5 + R(-6) <-- d_2 --
↦ R(-6)^9 + R(-7)^3 <-- d_3 --
↦ R(-7)^3 + R(-8)^3 <-- d_4 --
↦ R(-9), given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^5 + R(-6), given by a matrix, w\
  ith degrees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 ...
↦     -- -- -- -- -- -- -- -- +..
↦  0 : 4  4  5  5  5  5  5  6 |.1
↦     == == == == == == == ==
↦      4  4  5  5  5  5  5  6
↦ d_2 :
↦ Graded homomorphism: R(-4)^2 + R(-5)^5 + R(-6) <- R(-6)^9 + R(-7)^3, give\
  n by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   4 :  2   2   2   2   2   2   2   2   2   -   3   3 |..1
↦   4 :  2   2   2   2   2   2   2   2   2   -   3   3 |..2
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..3
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..4
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..5
↦   5 :  1   1   1   1   1   1   1   1   1   -   2   2 |..6
↦   5 :  1   1   1   1   1   1   1   1   1   -   2   2 |..7
↦   6 :  -   -   -   -   -   -   -   -   -   1   1   1 |..8
↦      === === === === === === === === === === === ===
↦        6   6   6   6   6   6   6   6   6   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^9 + R(-7)^3 <- R(-7)^3 + R(-8)^3, given by a m\
  atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   6 :  1   1   1   2   2   2 |..1
↦   6 :  1   1   1   2   2   2 |..2
↦   6 :  1   1   1   2   2   2 |..3
↦   6 :  1   1   1   2   2   2 |..4
↦   6 :  1   1   1   2   2   2 |..5
↦   6 :  1   1   1   2   2   2 |..6
↦   6 :  1   1   1   2   2   2 |..7
↦   6 :  1   1   1   2   2   2 |..8
↦   6 :  1   1   1   -   -   2 |..9
↦   7 :  -   -   -   1   1   1 |.10
↦   7 :  -   -   -   1   1   1 |.11
↦   7 :  -   -   -   1   1   1 |.12
↦      === === === === === ===
↦        7   7   7   8   8   8
↦ d_4 :
↦ Graded homomorphism: R(-7)^3 + R(-8)^3 <- R(-9), given by a matrix, with \
  degrees:
↦     .1 ...
↦     -- +..
↦  7 : 2 |.1
↦  7 : 2 |.2
↦  7 : 2 |.3
↦  8 : 1 |.4
↦  8 : 1 |.5
↦  8 : 1 |.6
↦     ==
↦      9
```

D.15.6.15 grtranspose1

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grtranspose1(M), graded object or list M

Return: same as input

Purpose: graded transpose of graded object or chain complex M

Assume: M must be a graded object or a list of graded objects

Example:

```
LIB "gradedModules.lib";
"Surface Name: 'k3.d10.g9.quart2' in P^4";
↦ Surface Name: 'k3.d10.g9.quart2' in P^4
int @p=31991; ring R = (@p),(x,y,z,u,v), dp;
ideal J = x3yz2+31/15x2y2z2-7231xy3z2+99/37y4z2+28/95x3z3+97/32x2yz3+13247xy2z3+12717
```

```
def I = grobj( groebner(J), intvec(0) ); // ASSUME: no zero entries in J!
ASSUME(0, grtest(I));
"Input degrees: "; grview(I);
↦ Input degrees:
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^5 + R(-6)^3, given by a matrix,\
    with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  4   4   5   5   5   5   5   6   6   6 |..1
↦      === === === === === === === === === ===
↦        4   4   5   5   5   5   5   6   6   6
def RR = grres(I, 0, 1); list L = RR;
" = Non-minimal betti numbers: "; print(betti(L, 0), "betti");
↦  = Non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     2     -     -     -
↦     4:     -     4     7     2     -
↦     5:     -     3     8     7     2
↦ ------------------------------------
↦ total:     1     9    15     9     2
↦ 
"Graded (original) structure of 'res(Input,0)': "; grview(L);
↦ Graded (original) structure of 'res(Input,0)':
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-4)^2 + R(-5)^4 + R(-6)^3 <-- d_2 --
↦ R(-7)^2 + R(-6)^7 + R(-7)^6 <-- d_3 --
↦ R(-7)^2 + R(-8)^7 <-- d_4 --
↦ R(-9)^2 <-- d_5 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^4 + R(-6)^3, given by a matrix,\
    with degrees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 .9 ...
↦     -- -- -- -- -- -- -- -- -- +..
↦  0 : 4  4  5  5  5  5  6  6  6 |.1
↦     == == == == == == == == ==
↦      4  4  5  5  5  5  6  6  6
↦ d_2 :
↦ Graded homomorphism: R(-4)^2 + R(-5)^4 + R(-6)^3 <- R(-7)^2 + R(-6)^7 + R\
   (-7)^6
↦ , given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   4 :  -   -   2   2   2   2   2   2   2   3   3   3   3   3   3 |..1
↦   4 :  -   -   2   2   2   2   2   2   2   3   3   3   3   3   3 |..2
↦   5 :  2   2   1   1   1   1   1   1   1   2   2   2   2   2   2 |..3
↦   5 :  2   2   1   1   1   1   1   1   1   2   2   2   2   2   2 |..4
↦   5 :  -   -   1   1   1   1   1   1   1   2   2   2   2   2   2 |..5
↦   5 :  -   -   1   1   1   1   1   1   1   2   2   2   2   2   2 |..6
↦   6 :  1   1   -   -   -   -   -   -   -   1   1   1   1   1   1 |..7
↦   6 :  1   1   -   -   -   -   -   -   -   1   1   1   1   1   1 |..8
↦   6 :  1   1   -   -   -   -   -   -   -   1   1   1   1   1   1 |..9
↦      === === === === === === === === === === === === === === ===
↦        7   7   6   6   6   6   6   6   6   7   7   7   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-7)^2 + R(-6)^7 + R(-7)^6 <- R(-7)^2 + R(-8)^7, gi\
  ven by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦      --- --- --- --- --- --- --- --- --- +...
↦   7 :  -   -   1   1   1   1   1   1   1 |..1
↦   7 :  -   -   1   1   1   1   1   1   1 |..2
↦   6 :  1   1   2   2   2   2   2   2   2 |..3
↦   6 :  1   1   2   2   2   2   2   2   2 |..4
↦   6 :  1   1   2   2   2   2   2   2   2 |..5
↦   6 :  1   1   2   2   2   2   2   2   2 |..6
↦   6 :  1   1   2   2   2   2   2   2   2 |..7
↦   6 :  1   1   2   2   2   2   2   2   2 |..8
↦   6 :  1   1   2   2   2   2   2   2   2 |..9
↦   7 :  -   -   1   1   1   1   1   1   1 |.10
↦   7 :  -   -   1   1   1   1   1   1   1 |.11
↦   7 :  -   -   1   1   1   1   1   1   1 |.12
↦   7 :  -   -   1   1   1   1   1   1   1 |.13
↦   7 :  -   -   1   1   1   1   1   1   1 |.14
↦   7 :  -   -   1   1   1   1   1   1   1 |.15
↦      === === === === === === === === ===
↦        7   7   8   8   8   8   8   8   8
↦ d_4 :
↦ Graded homomorphism: R(-7)^2 + R(-8)^7 <- R(-9)^2, given by a matrix, wit\
  h degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  7 :  2  2 |.1
↦  7 :  2  2 |.2
↦  8 :  1  1 |.3
↦  8 :  1  1 |.4
↦  8 :  1  1 |.5
↦  8 :  1  1 |.6
↦  8 :  1  1 |.7
↦  8 :  1  1 |.8
↦  8 :  1  1 |.9
↦     == ==
↦      9  9
↦ d_5 :
↦ Graded homomorphism: R(-9)^2 <- 0, given by zero (2 x 0) matrix.
"Graded transpose of the previous resolution "; list LLL = grtranspose1( L ); grview
↦ Graded transpose of the previous resolution
↦ Graded resolution:
↦ R(9)^2 <-- d_1 --
↦ R(8)^7 + R(7)^2 <-- d_2 --
↦ R(7)^8 + R(6)^7 <-- d_3 --
↦ R(6)^3 + R(5)^4 + R(4)^2 <-- d_4 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(9)^2 <- R(8)^7 + R(7)^2, given by a matrix, with d\
  egrees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦       --- --- --- --- --- --- --- --- --- +...
↦  -9 :  1   1   1   1   1   1   1   2   2 |..1
↦  -9 :  1   1   1   1   1   1   1   2   2 |..2
↦       === === === === === === === === ===
↦        -8  -8  -8  -8  -8  -8  -8  -7  -7
↦ d_2 :
↦ Graded homomorphism: R(8)^7 + R(7)^2 <- R(7)^8 + R(6)^7, given by a matri\
  x, with degrees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦       --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..1
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..2
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..3
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..4
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..5
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..6
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..7
↦  -7 :  -   -   -   -   -   -   -   -   1   1   1   1   1   1   1 |..8
↦  -7 :  -   -   -   -   -   -   -   -   1   1   1   1   1   1   1 |..9
↦       === === === === === === === === === === === === === === ===
↦        -7  -7  -7  -7  -7  -7  -7  -7  -6  -6  -6  -6  -6  -6  -6
↦ d_3 :
↦ Graded homomorphism: R(7)^8 + R(6)^7 <- R(6)^3 + R(5)^4 + R(4)^2, given b\
  y a matrix, with degrees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦       --- --- --- --- --- --- --- --- --- +...
↦  -7 :  1   1   1   2   2   -   -   -   - |..1
↦  -7 :  1   1   1   2   2   -   -   -   - |..2
↦  -7 :  1   1   1   2   2   2   2   3   3 |..3
↦  -7 :  1   1   1   2   2   2   2   3   3 |..4
↦  -7 :  1   1   1   2   2   2   2   3   3 |..5
↦  -7 :  1   1   1   2   2   2   2   3   3 |..6
↦  -7 :  1   1   1   2   2   2   2   3   3 |..7
↦  -7 :  1   1   1   2   2   2   2   3   3 |..8
↦  -6 :  -   -   -   1   1   1   1   2   2 |..9
↦  -6 :  -   -   -   1   1   1   1   2   2 |.10
↦  -6 :  -   -   -   1   1   1   1   2   2 |.11
↦  -6 :  -   -   -   1   1   1   1   2   2 |.12
↦  -6 :  -   -   -   1   1   1   1   2   2 |.13
↦  -6 :  -   -   -   1   1   1   1   2   2 |.14
↦  -6 :  -   -   -   1   1   1   1   2   2 |.15
↦       === === === === === === === === ===
↦        -6  -6  -6  -5  -5  -5  -5  -4  -4
↦ d_4 :
↦ Graded homomorphism: R(6)^3 + R(5)^4 + R(4)^2 <- R, given by a matrix, wi\
  th degrees:
↦       ..1 ....
↦       --- +...
↦  -6 :  6 |..1
↦  -6 :  6 |..2
↦  -6 :  6 |..3
↦  -5 :  5 |..4
↦  -5 :  5 |..5
↦  -5 :  5 |..6
↦  -5 :  5 |..7
↦  -4 :  4 |..8
↦  -4 :  4 |..9
↦      ===
↦        0
"Its non-minimal betti numbers: "; print(betti(LLL, 0), "betti");
↦ Its non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦    -9:     2     7     8     3     -
↦    -8:     -     2     7     4     -
↦    -7:     -     -     -     2     -
↦    -6:     -     -     -     -     -
↦    -5:     -     -     -     -     -
↦    -4:     -     -     -     -     1
↦ ------------------------------------
↦ total:     2     9    15     9     1
↦
```

D.15.6.16 TestGRRes

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: TestGRRes(name, I), string name, ideal I

Return: nothing

Purpose: compute/test/output/order/transpose a graded resolution of I

Example:

```
LIB "gradedModules.lib";
//  if( defined(assumeLevel) ){ int assumeLevel0 = assumeLevel; } else { int assumeLe
// note: data from random generation 2
string Name = "castelnuovo"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = 51
↦ ==============================================
↦
↦ === Example: [ castelnuovo ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     1     -
↦     2:     -     2     2
↦ ------------------------
↦ total:     1     3     2
↦
↦  = Degrees of (ordered) maps:
```

```
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-2) + R(-3)^2 <-- d_2 --
↦ R(-4)^2, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-2) + R(-3)^2, given by a matrix, with degree\
  s:
↦     .1 .2 .3 ...
↦     -- -- -- +..
↦  0 : 2  3  3 |.1
↦     == == ==
↦      2  3  3
↦ d_2 :
↦ Graded homomorphism: R(-2) + R(-3)^2 <- R(-4)^2, given by a matrix, with \
  degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  2 : 2  2 |.1
↦  3 : 1  1 |.2
↦  3 : 1  1 |.3
↦     == ==
↦      4  4
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2
↦ ------------------------
↦    -4:     2     2     -
↦    -3:     -     1     -
↦    -2:     -     -     1
↦ ------------------------
↦ total:     2     3     1
↦ 
↦ Graded resolution:
↦ R(4)^2 <-- d_1 --
↦ R(3)^2 + R(2) <-- d_2 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(4)^2 <- R(3)^2 + R(2), given by a matrix, with deg\
  rees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦  -4 :  1   1   2 |..1
↦  -4 :  1   1   2 |..2
↦      === === ===
↦       -3  -3  -2
↦ d_2 :
↦ Graded homomorphism: R(3)^2 + R(2) <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -3 :  3 |..1
↦  -3 :  3 |..2
↦  -2 :  2 |..3
↦      ===
↦        0
```

```
↦
↦
string Name = "ell.d8.g7"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = x2y
↦ ==============================================
↦
↦ === Example: [ ell.d8.g7 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     -     -
↦     2:     -     2     -
↦     3:     -     1     2
↦ ------------------------
↦ total:     1     3     2
↦
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-3)^2 + R(-4) <-- d_2 --
↦ R(-5)^2, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-3)^2 + R(-4), given by a matrix, with degree\
  s:
↦     .1 .2 .3 ...
↦     -- -- -- +..
↦  0 : 3  3  4 |.1
↦     == == ==
↦      3  3  4
↦ d_2 :
↦ Graded homomorphism: R(-3)^2 + R(-4) <- R(-5)^2, given by a matrix, with \
  degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  3 : 2  2 |.1
↦  3 : 2  2 |.2
↦  4 : 1  1 |.3
↦     == ==
↦      5  5
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%
↦            0     1     2
↦ ------------------------
↦    -5:     2     1     -
↦    -4:     -     2     -
↦    -3:     -     -     -
↦    -2:     -     -     1
↦ ------------------------
↦ total:     2     3     1
↦
↦ Graded resolution:
↦ R(5)^2 <-- d_1 --
↦ R(4) + R(3)^2 <-- d_2 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(5)^2 <- R(4) + R(3)^2, given by a matrix, with deg\
  rees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦  -5 :  1   2   2 |..1
↦  -5 :  1   2   2 |..2
↦      === === ===
↦       -4  -3  -3
↦ d_2 :
↦ Graded homomorphism: R(4) + R(3)^2 <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -4 :  4 |..1
↦  -3 :  3 |..2
↦  -3 :  3 |..3
↦      ===
↦        0
↦ 
↦ 
string Name = "ell.d7.g6"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = 4971
↦ ==============================================
↦ 
↦ === Example: [ ell.d7.g6 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     1     -
↦     2:     -     -     -
↦     3:     -     2     2
↦ ------------------------
↦ total:     1     3     2
↦ 
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-2) + R(-4)^2 <-- d_2 --
↦ R(-5)^2, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-2) + R(-4)^2, given by a matrix, with degree\
  s:
↦     .1 .2 .3 ...
↦     -- -- -- +..
↦  0 : 2  4  4 |.1
↦     == == ==
↦      2  4  4
↦ d_2 :
↦ Graded homomorphism: R(-2) + R(-4)^2 <- R(-5)^2, given by a matrix, with \
  degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  2 : 3  3 |.1
↦  4 : 1  1 |.2
↦  4 : 1  1 |.3
↦     == ==
↦      5  5
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2
↦ ------------------------
↦    -5:     2     2     -
↦    -4:     -     -     -
↦    -3:     -     1     -
↦    -2:     -     -     1
↦ ------------------------
↦ total:     2     3     1
↦
↦ Graded resolution:
↦ R(5)^2 <-- d_1 --
↦ R(4)^2 + R(2) <-- d_2 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(5)^2 <- R(4)^2 + R(2), given by a matrix, with deg\
  rees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦  -5 :  1   1   3 |..1
↦  -5 :  1   1   3 |..2
↦      === === ===
↦       -4  -4  -2
↦ d_2 :
↦ Graded homomorphism: R(4)^2 + R(2) <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -4 :  4 |..1
↦  -4 :  4 |..2
↦  -2 :  2 |..3
↦      ===
↦        0
↦
↦
string Name = "k3.d7.g5"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = -97/
↦ ==============================================
↦
↦ === Example: [ k3.d7.g5 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2
↦ ------------------------
↦     0:     1     -     -
↦     1:     -     -     -
↦     2:      -     3     1
↦     3:      -     -     1
↦ ------------------------
↦ total:      1     3     2
↦
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-3)^3 <-- d_2 --
↦ R(-4) + R(-5), given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-3)^3, given by a matrix, with degrees:
↦     .1 .2 .3 ...
↦     -- -- -- +..
↦  0 : 3  3  3 |.1
↦     == == ==
↦      3  3  3
↦ d_2 :
↦ Graded homomorphism: R(-3)^3 <- R(-4) + R(-5), given by a matrix, with de\
  grees:
↦     .1 .2 ...
↦     -- -- +..
↦  3 : 1  2 |.1
↦  3 : 1  2 |.2
↦  3 : 1  2 |.3
↦     == ==
↦      4  5
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦             0     1     2
↦ ------------------------
↦    -5:      1     -     -
↦    -4:      1     3     -
↦    -3:      -     -     -
↦    -2:      -     -     1
↦ ------------------------
↦ total:      2     3     1
↦
↦ Graded resolution:
↦ R(5) + R(4) <-- d_1 --
↦ R(3)^3 <-- d_2 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(5) + R(4) <- R(3)^3, given by a matrix, with degre\
  es:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦  -5 :  2   2   2 |..1
↦  -4 :  1   1   1 |..2
↦      === === ===
↦       -3  -3  -3
↦ d_2 :
↦ Graded homomorphism: R(3)^3 <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -3 :  3 |..1
↦  -3 :  3 |..2
↦  -3 :  3 |..3
↦      ===
↦        0
↦ 
↦ 
string Name = "rat.d8.g6"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = -19/
↦ ==============================================
↦ 
↦ === Example: [ rat.d8.g6 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     -     -     -
↦     2:     -     1     -     -
↦     3:     -     4     5     1
↦ ------------------------------
↦ total:     1     5     5     1
↦ 
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-3) + R(-4)^4 <-- d_2 --
↦ R(-5)^5 <-- d_3 --
↦ R(-6), given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-3) + R(-4)^4, given by a matrix, with degree\
   s:
↦     .1 .2 .3 .4 .5 ...
↦     -- -- -- -- -- +..
↦  0 : 3  4  4  4  4 |.1
↦     == == == == ==
↦      3  4  4  4  4
↦ d_2 :
↦ Graded homomorphism: R(-3) + R(-4)^4 <- R(-5)^5, given by a square matrix\
   , with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   3 :  -   2   2   2   2 |..1
↦   4 :  1   1   1   1   1 |..2
↦   4 :  1   1   1   1   1 |..3
↦   4 :  1   1   1   1   1 |..4
↦   4 :  1   1   1   1   1 |..5
↦      === === === === ===
↦        5   5   5   5   5
↦ d_3 :
↦ Graded homomorphism: R(-5)^5 <- R(-6), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  5 : 1 |.1
↦  5 : 1 |.2
↦  5 : 1 |.3
↦  5 : 1 |.4
↦  5 : 1 |.5
↦     ==
↦      6
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2     3
↦ ------------------------------
↦    -6:     1     5     4     -
↦    -5:     -     -     1     -
↦    -4:     -     -     -     -
↦    -3:     -     -     -     1
↦ ------------------------------
↦ total:     1     5     5     1
↦ 
↦ Graded resolution:
↦ R(6) <-- d_1 --
↦ R(5)^5 <-- d_2 --
↦ R(4)^4 + R(3) <-- d_3 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(6) <- R(5)^5, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦  -6 :  1   1   1   1   1 |..1
↦      === === === === ===
↦       -5  -5  -5  -5  -5
↦ d_2 :
↦ Graded homomorphism: R(5)^5 <- R(4)^4 + R(3), given by a square matrix, w\
  ith degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦  -5 :  1   1   1   1   - |..1
↦  -5 :  1   1   1   1   2 |..2
↦  -5 :  1   1   1   1   2 |..3
↦  -5 :  1   1   1   1   2 |..4
↦  -5 :  1   1   1   1   2 |..5
↦      === === === === ===
↦       -4  -4  -4  -4  -3
↦ d_3 :
↦ Graded homomorphism: R(4)^4 + R(3) <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -4 :  4 |..1
↦  -4 :  4 |..2
↦  -4 :  4 |..3
↦  -4 :  4 |..4
↦  -3 :  3 |..5
↦      ===
↦        0
```

```
↦
↦
string Name = "k3.d14.g19"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = x4y
↦ ==============================================
↦
↦ === Example: [ k3.d14.g19 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:     -     -     -     -
↦     2:     -     -     -     -
↦     3:     -     -     -     -
↦     4:     -     4     2     -
↦     5:     -     4     8     3
↦ ------------------------------
↦ total:     1     8    10     3
↦
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-5)^4 + R(-6)^4 <-- d_2 --
↦ R(-6)^2 + R(-7)^8 <-- d_3 --
↦ R(-8)^3, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-5)^4 + R(-6)^4, given by a matrix, with degr\
   ees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 ...
↦     -- -- -- -- -- -- -- -- +..
↦  0 : 5  5  5  5  6  6  6  6 |.1
↦     == == == == == == == ==
↦      5  5  5  5  6  6  6  6
↦ d_2 :
↦ Graded homomorphism: R(-5)^4 + R(-6)^4 <- R(-6)^2 + R(-7)^8, given by a m\
   atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   5 :  1   1   2   2   2   2   2   2   2   2 |..1
↦   5 :  1   1   2   2   2   2   2   2   2   2 |..2
↦   5 :  1   1   2   2   2   2   2   2   2   2 |..3
↦   5 :  1   1   2   2   2   2   2   2   2   2 |..4
↦   6 :  -   -   1   1   1   1   1   1   1   1 |..5
↦   6 :  -   -   1   1   1   1   1   1   1   1 |..6
↦   6 :  -   -   1   1   1   1   1   1   1   1 |..7
↦   6 :  -   -   1   1   1   1   1   1   1   1 |..8
↦      === === === === === === === === === ===
↦        6   6   7   7   7   7   7   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^2 + R(-7)^8 <- R(-8)^3, given by a matrix, wit\
   h degrees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦    6 :  2   2   2 |..1
↦    6 :  2   2   2 |..2
↦    7 :  1   1   1 |..3
↦    7 :  1   1   1 |..4
↦    7 :  1   1   1 |..5
↦    7 :  1   1   1 |..6
↦    7 :  1   1   1 |..7
↦    7 :  1   1   1 |..8
↦    7 :  1   1   1 |..9
↦    7 :  1   1   1 |.10
↦      === === ===
↦        8   8   8
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2     3
↦ ------------------------------
↦    -8:     3     8     4     -
↦    -7:     -     2     4     -
↦    -6:     -     -     -     -
↦    -5:     -     -     -     -
↦    -4:     -     -     -     -
↦    -3:     -     -     -     1
↦ ------------------------------
↦ total:     3    10     8     1
↦ 
↦ Graded resolution:
↦ R(8)^3 <-- d_1 --
↦ R(7)^8 + R(6)^2 <-- d_2 --
↦ R(6)^4 + R(5)^4 <-- d_3 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(8)^3 <- R(7)^8 + R(6)^2, given by a matrix, with d\
  egrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   1   1   1   1   1   1   1   2   2 |..1
↦  -8 :  1   1   1   1   1   1   1   1   2   2 |..2
↦  -8 :  1   1   1   1   1   1   1   1   2   2 |..3
↦      === === === === === === === === === ===
↦       -7  -7  -7  -7  -7  -7  -7  -7  -6  -6
↦ d_2 :
↦ Graded homomorphism: R(7)^8 + R(6)^2 <- R(6)^4 + R(5)^4, given by a matri\
  x, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ....
↦      --- --- --- --- --- --- --- --- +...
↦  -7 :  1   1   1   1   2   2   2   2 |..1
↦  -7 :  1   1   1   1   2   2   2   2 |..2
↦  -7 :  1   1   1   1   2   2   2   2 |..3
↦  -7 :  1   1   1   1   2   2   2   2 |..4
↦  -7 :  1   1   1   1   2   2   2   2 |..5
↦  -7 :  1   1   1   1   2   2   2   2 |..6
↦  -7 :  1   1   1   1   2   2   2   2 |..7
↦  -7 :  1   1   1   1   2   2   2   2 |..8
↦  -6 :  -   -   -   -   1   1   1   1 |..9
↦  -6 :  -   -   -   -   1   1   1   1 |.10
↦      === === === === === === === ===
↦       -6  -6  -6  -6  -5  -5  -5  -5
↦ d_3 :
↦ Graded homomorphism: R(6)^4 + R(5)^4 <- R, given by a matrix, with degree\
  s:
↦      ..1 ....
↦      --- +...
↦  -6 :  6 |..1
↦  -6 :  6 |..2
↦  -6 :  6 |..3
↦  -6 :  6 |..4
↦  -5 :  5 |..5
↦  -5 :  5 |..6
↦  -5 :  5 |..7
↦  -5 :  5 |..8
↦      ===
↦        0
↦ 
↦ 
string Name = "k3.d11.g11.ss0"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I =
↦ ==============================================
↦ 
↦ === Example: [ k3.d11.g11.ss0 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     -     -     -     -
↦     4:     -     9     8     -     -
↦     5:     -     -     5     7     2
↦ ------------------------------------
↦ total:     1     9    13     7     2
↦ 
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-5)^9 <-- d_2 --
↦ R(-6)^8 + R(-7)^5 <-- d_3 --
↦ R(-8)^7 <-- d_4 --
↦ R(-9)^2, given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-5)^9, given by a matrix, with degrees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 .9 ...
↦     -- -- -- -- -- -- -- -- -- +..
↦  0 : 5  5  5  5  5  5  5  5  5 |.1
↦     == == == == == == == == ==
↦      5  5  5  5  5  5  5  5  5
↦ d_2 :
↦ Graded homomorphism: R(-5)^9 <- R(-6)^8 + R(-7)^5, given by a matrix, wit\
  h degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..1
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..2
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..3
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..4
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..5
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..6
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..7
↦   5 :  1   1   1   1   1   1   1   1   2   2   2   2   2 |..8
↦   5 :  1   1   1   1   1   1   1   1   -   -   -   -   - |..9
↦      === === === === === === === === === === === === ===
↦        6   6   6   6   6   6   6   6   7   7   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^8 + R(-7)^5 <- R(-8)^7, given by a matrix, wit\
  h degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ....
↦      --- --- --- --- --- --- --- +...
↦   6 :  -   -   2   2   2   2   2 |..1
↦   6 :  -   -   2   2   2   2   2 |..2
↦   6 :  -   -   2   2   2   2   2 |..3
↦   6 :  -   -   2   2   2   2   2 |..4
↦   6 :  -   -   2   2   2   2   2 |..5
↦   6 :  -   -   2   2   2   2   2 |..6
↦   6 :  -   -   2   2   2   2   2 |..7
↦   6 :  -   -   2   2   2   2   2 |..8
↦   7 :  1   1   1   1   1   1   1 |..9
↦   7 :  1   1   1   1   1   1   1 |.10
↦   7 :  1   1   1   1   1   1   1 |.11
↦   7 :  1   1   1   1   1   1   1 |.12
↦   7 :  1   1   1   1   1   1   1 |.13
↦      === === === === === === ===
↦        8   8   8   8   8   8   8
↦ d_4 :
↦ Graded homomorphism: R(-8)^7 <- R(-9)^2, given by a matrix, with degrees:
↦     .1 .2 ...
↦     -- -- +..
↦  8 : 1  1 |.1
↦  8 : 1  1 |.2
↦  8 : 1  1 |.3
↦  8 : 1  1 |.4
↦  8 : 1  1 |.5
↦  8 : 1  1 |.6
↦  8 : 1  1 |.7
↦     == ==
↦      9  9
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2     3     4
↦ ------------------------------------
↦    -9:     2     7     5     -     -
↦    -8:     -     -     8     9     -
↦    -7:     -     -     -     -     -
↦    -6:     -     -     -     -     -
↦    -5:     -     -     -     -     -
↦    -4:     -     -     -     -     1
↦ ------------------------------------
↦ total:     2     7    13     9     1
↦ 
↦ Graded resolution:
↦ R(9)^2 <-- d_1 --
↦ R(8)^7 <-- d_2 --
↦ R(7)^5 + R(6)^8 <-- d_3 --
↦ R(5)^9 <-- d_4 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(9)^2 <- R(8)^7, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ....
↦      --- --- --- --- --- --- --- +...
↦  -9 :  1   1   1   1   1   1   1 |..1
↦  -9 :  1   1   1   1   1   1   1 |..2
↦      === === === === === === ===
↦       -8  -8  -8  -8  -8  -8  -8
↦ d_2 :
↦ Graded homomorphism: R(8)^7 <- R(7)^5 + R(6)^8, given by a matrix, with d\
  egrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   1   1   1   1   -   -   -   -   -   -   -   - |..1
↦  -8 :  1   1   1   1   1   -   -   -   -   -   -   -   - |..2
↦  -8 :  1   1   1   1   1   2   2   2   2   2   2   2   2 |..3
↦  -8 :  1   1   1   1   1   2   2   2   2   2   2   2   2 |..4
↦  -8 :  1   1   1   1   1   2   2   2   2   2   2   2   2 |..5
↦  -8 :  1   1   1   1   1   2   2   2   2   2   2   2   2 |..6
↦  -8 :  1   1   1   1   1   2   2   2   2   2   2   2   2 |..7
↦      === === === === === === === === === === === === ===
↦       -7  -7  -7  -7  -7  -6  -6  -6  -6  -6  -6  -6  -6
↦ d_3 :
↦ Graded homomorphism: R(7)^5 + R(6)^8 <- R(5)^9, given by a matrix, with d\
  egrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦      --- --- --- --- --- --- --- --- --- +...
↦  -7 :  2   2   2   2   2   2   2   2   - |..1
↦  -7 :  2   2   2   2   2   2   2   2   - |..2
↦  -7 :  2   2   2   2   2   2   2   2   - |..3
↦  -7 :  2   2   2   2   2   2   2   2   - |..4
↦  -7 :  2   2   2   2   2   2   2   2   - |..5
↦  -6 :  1   1   1   1   1   1   1   1   1 |..6
↦  -6 :  1   1   1   1   1   1   1   1   1 |..7
↦  -6 :  1   1   1   1   1   1   1   1   1 |..8
↦  -6 :  1   1   1   1   1   1   1   1   1 |..9
↦  -6 :  1   1   1   1   1   1   1   1   1 |.10
↦  -6 :  1   1   1   1   1   1   1   1   1 |.11
↦  -6 :  1   1   1   1   1   1   1   1   1 |.12
↦  -6 :  1   1   1   1   1   1   1   1   1 |.13
↦      === === === === === === === === ===
↦       -5  -5  -5  -5  -5  -5  -5  -5  -5
↦ d_4 :
↦ Graded homomorphism: R(5)^9 <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -5 :  5 |..1
↦  -5 :  5 |..2
↦  -5 :  5 |..3
↦  -5 :  5 |..4
↦  -5 :  5 |..5
↦  -5 :  5 |..6
↦  -5 :  5 |..7
↦  -5 :  5 |..8
↦  -5 :  5 |..9
↦      ===
↦        0
↦ 
↦ 
string Name = "ell.d10.g9"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I = -44
↦ ==============================================
↦ 
↦ === Example: [ ell.d10.g9 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via ’grres’:
↦  = Non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     1     -     -     -
↦     4:     -     9    14     5     -
↦     5:     -     -     1     2     1
↦ ------------------------------------
↦ total:     1    10    15     7     1
↦ 
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-4) + R(-5)^9 <-- d_2 --
↦ R(-6)^14 + R(-7) <-- d_3 --
↦ R(-7)^5 + R(-8)^2 <-- d_4 --
↦ R(-9), given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-4) + R(-5)^9, given by a matrix, with degree\
  s:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  4   5   5   5   5   5   5   5   5   5 |..1
↦      === === === === === === === === === ===
↦        4   5   5   5   5   5   5   5   5   5
↦ d_2 :
↦ Graded homomorphism: R(-4) + R(-5)^9 <- R(-6)^14 + R(-7), given by a matr\
  ix, with degrees:
↦        ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦        --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦    4 :   -   -   -   -   -   2   2   2   2   2   2   2   2   2   - |..1
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2 |..2
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2 |..3
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2 |..4
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2 |..5
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2 |..6
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   - |..7
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   - |..8
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   - |..9
↦    5 :   1   1   1   1   1   1   1   1   1   1   1   1   1   1   - |.10
↦        === === === === === === === === === === === === === === ===
↦          6   6   6   6   6   6   6   6   6   6   6   6   6   6   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^14 + R(-7) <- R(-7)^5 + R(-8)^2, given by a ma\
  trix, with degrees:
↦        ..1 ..2 ..3 ..4 ..5 ..6 ..7 ....
↦        --- --- --- --- --- --- --- +...
↦    6 :   1   1   1   1   1   2   - |..1
↦    6 :   1   1   1   1   1   2   - |..2
↦    6 :   1   1   1   1   1   2   - |..3
↦    6 :   1   1   1   1   1   2   - |..4
↦    6 :   1   1   1   1   1   2   - |..5
↦    6 :   1   1   1   1   1   -   2 |..6
↦    6 :   1   1   1   1   1   -   2 |..7
↦    6 :   1   1   1   1   1   -   2 |..8
↦    6 :   1   1   1   1   1   -   2 |..9
↦    6 :   1   1   1   1   1   -   2 |.10
↦    6 :   1   1   1   1   1   -   2 |.11
↦    6 :   1   1   1   1   1   -   2 |.12
↦    6 :   1   1   1   1   1   -   2 |.13
↦    6 :   1   1   1   1   1   -   - |.14
↦    7 :   -   -   -   -   -   1   1 |.15
↦        === === === === === === ===
↦          7   7   7   7   7   8   8
↦ d_4 :
↦ Graded homomorphism: R(-7)^5 + R(-8)^2 <- R(-9), given by a matrix, with \
  degrees:
↦       .1 ...
↦       -- +..
↦   7 :  2 |.1
↦   7 :  2 |.2
↦   7 :  2 |.3
↦   7 :  2 |.4
↦   7 :  2 |.5
↦   8 :  1 |.6
↦   8 :  1 |.7
↦       ==
↦        9
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%
↦            0     1     2     3     4
↦ ------------------------------------
↦    -9:     1     2     1     -     -
↦    -8:     -     5    14     9     -
↦    -7:     -     -     -     1     -
↦    -6:     -     -     -     -     -
↦    -5:     -     -     -     -     -
↦    -4:     -     -     -     -     1
↦ ------------------------------------
↦ total:     1     7    15    10     1
↦ 
↦ Graded resolution:
↦ R(9) <-- d_1 --
↦ R(8)^2 + R(7)^5 <-- d_2 --
↦ R(7) + R(6)^14 <-- d_3 --
↦ R(5)^9 + R(4) <-- d_4 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(9) <- R(8)^2 + R(7)^5, given by a matrix, with deg\
  rees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ....
↦      --- --- --- --- --- --- --- +...
↦  -9 :  1   1   2   2   2   2   2 |..1
↦      === === === === === === ===
↦       -8  -8  -7  -7  -7  -7  -7
↦ d_2 :
↦ Graded homomorphism: R(8)^2 + R(7)^5 <- R(7) + R(6)^14, given by a matrix\
  , with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   2   2   2   2   2   -   -   -   -   -   -   -   -   - |..1
↦  -8 :  1   -   -   -   -   -   2   2   2   2   2   2   2   2   - |..2
↦  -7 :  -   1   1   1   1   1   1   1   1   1   1   1   1   1   1 |..3
↦  -7 :  -   1   1   1   1   1   1   1   1   1   1   1   1   1   1 |..4
↦  -7 :  -   1   1   1   1   1   1   1   1   1   1   1   1   1   1 |..5
↦  -7 :  -   1   1   1   1   1   1   1   1   1   1   1   1   1   1 |..6
↦  -7 :  -   1   1   1   1   1   1   1   1   1   1   1   1   1   1 |..7
↦      === === === === === === === === === === === === === === ===
↦       -7  -6  -6  -6  -6  -6  -6  -6  -6  -6  -6  -6  -6  -6  -6
↦ d_3 :
↦ Graded homomorphism: R(7) + R(6)^14 <- R(5)^9 + R(4), given by a matrix, \
  with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦  -7 :  2   2   2   2   2   -   -   -   -   - |..1
↦  -6 :  1   1   1   1   1   1   1   1   1   - |..2
↦  -6 :  1   1   1   1   1   1   1   1   1   - |..3
↦  -6 :  1   1   1   1   1   1   1   1   1   - |..4
↦  -6 :  1   1   1   1   1   1   1   1   1   - |..5
↦  -6 :  1   1   1   1   1   1   1   1   1   - |..6
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |..7
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |..8
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |..9
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.10
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.11
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.12
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.13
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.14
↦  -6 :  1   1   1   1   1   1   1   1   1   2 |.15
↦      === === === === === === === === === ===
↦       -5  -5  -5  -5  -5  -5  -5  -5  -5  -4
↦ d_4 :
↦ Graded homomorphism: R(5)^9 + R(4) <- R, given by a matrix, with degrees:
↦      ..1 ....
↦      --- +...
↦  -5 :  5 |..1
↦  -5 :  5 |..2
↦  -5 :  5 |..3
↦  -5 :  5 |..4
↦  -5 :  5 |..5
↦  -5 :  5 |..6
↦  -5 :  5 |..7
↦  -5 :  5 |..8
↦  -5 :  5 |..9
↦  -4 :  4 |.10
↦      ===
↦        0
↦
↦
string Name = "k3.d10.g9.quart2"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal I
↦ =============================================
↦
↦ === Example: [ k3.d10.g9.quart2 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     2     -     -     -
↦     4:     -     4     7     2     -
↦     5:     -     3     8     7     2
↦ ------------------------------------
↦ total:     1     9    15     9     2
↦
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-4)^2 + R(-5)^4 + R(-6)^3 <-- d_2 --
↦ R(-6)^7 + R(-7)^8 <-- d_3 --
↦ R(-7)^2 + R(-8)^7 <-- d_4 --
↦ R(-9)^2, given by maps:
↦ d_1 :
```

```
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^4 + R(-6)^3, given by a matrix,\
   with degrees:
↦       .1 .2 .3 .4 .5 .6 .7 .8 .9 ...
↦       -- -- -- -- -- -- -- -- -- +..
↦    0 : 4  4  5  5  5  5  6  6  6 |.1
↦       == == == == == == == == ==
↦        4  4  5  5  5  5  6  6  6
↦ d_2 :
↦ Graded homomorphism: R(-4)^2 + R(-5)^4 + R(-6)^3 <- R(-6)^7 + R(-7)^8, gi\
  ven by a matrix, with degrees:
↦         ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦         --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦      4 :  2   2   2   2   2   2   2   -   -   3   3   3   3   3   3 |..1
↦      4 :  2   2   2   2   2   2   2   -   -   3   3   3   3   3   3 |..2
↦      5 :  1   1   1   1   1   1   1   2   2   2   2   2   2   2   2 |..3
↦      5 :  1   1   1   1   1   1   1   2   2   2   2   2   2   2   2 |..4
↦      5 :  1   1   1   1   1   1   1   -   -   2   2   2   2   2   2 |..5
↦      5 :  1   1   1   1   1   1   1   -   -   2   2   2   2   2   2 |..6
↦      6 :  -   -   -   -   -   -   -   1   1   1   1   1   1   1   1 |..7
↦      6 :  -   -   -   -   -   -   -   1   1   1   1   1   1   1   1 |..8
↦      6 :  -   -   -   -   -   -   -   1   1   1   1   1   1   1   1 |..9
↦         === === === === === === === === === === === === === === ===
↦           6   6   6   6   6   6   6   7   7   7   7   7   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^7 + R(-7)^8 <- R(-7)^2 + R(-8)^7, given by a m\
  atrix, with degrees:
↦         ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦         --- --- --- --- --- --- --- --- --- +...
↦      6 :  1   1   2   2   2   2   2   2   2 |..1
↦      6 :  1   1   2   2   2   2   2   2   2 |..2
↦      6 :  1   1   2   2   2   2   2   2   2 |..3
↦      6 :  1   1   2   2   2   2   2   2   2 |..4
↦      6 :  1   1   2   2   2   2   2   2   2 |..5
↦      6 :  1   1   2   2   2   2   2   2   2 |..6
↦      6 :  1   1   2   2   2   2   2   2   2 |..7
↦      7 :  -   -   1   1   1   1   1   1   1 |..8
↦      7 :  -   -   1   1   1   1   1   1   1 |..9
↦      7 :  -   -   1   1   1   1   1   1   1 |.10
↦      7 :  -   -   1   1   1   1   1   1   1 |.11
↦      7 :  -   -   1   1   1   1   1   1   1 |.12
↦      7 :  -   -   1   1   1   1   1   1   1 |.13
↦      7 :  -   -   1   1   1   1   1   1   1 |.14
↦      7 :  -   -   1   1   1   1   1   1   1 |.15
↦         === === === === === === === === ===
↦           7   7   8   8   8   8   8   8   8
↦ d_4 :
↦ Graded homomorphism: R(-7)^2 + R(-8)^7 <- R(-9)^2, given by a matrix, wit\
  h degrees:
↦       .1 .2 ...
↦       -- -- +..
↦    7 : 2  2 |.1
↦    7 : 2  2 |.2
↦    8 : 1  1 |.3
↦  8 : 1  1 |.4
↦  8 : 1  1 |.5
↦  8 : 1  1 |.6
↦  8 : 1  1 |.7
↦  8 : 1  1 |.8
↦  8 : 1  1 |.9
↦     == ==
↦      9  9
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%%
↦            0     1     2     3     4
↦ ------------------------------------
↦    -9:     2     7     8     3     -
↦    -8:     -     2     7     4     -
↦    -7:     -     -     -     2     -
↦    -6:     -     -     -     -     -
↦    -5:     -     -     -     -     -
↦    -4:     -     -     -     -     1
↦ ------------------------------------
↦ total:     2     9    15     9     1
↦ 
↦ Graded resolution:
↦ R(9)^2 <-- d_1 --
↦ R(8)^7 + R(7)^2 <-- d_2 --
↦ R(7)^8 + R(6)^7 <-- d_3 --
↦ R(6)^3 + R(5)^4 + R(4)^2 <-- d_4 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(9)^2 <- R(8)^7 + R(7)^2, given by a matrix, with d\
  egrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦      --- --- --- --- --- --- --- --- --- +...
↦  -9 :  1   1   1   1   1   1   1   2   2 |..1
↦  -9 :  1   1   1   1   1   1   1   2   2 |..2
↦      === === === === === === === === ===
↦       -8  -8  -8  -8  -8  -8  -8  -7  -7
↦ d_2 :
↦ Graded homomorphism: R(8)^7 + R(7)^2 <- R(7)^8 + R(6)^7, given by a matri\
  x, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..1
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..2
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..3
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..4
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..5
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..6
↦  -8 :  1   1   1   1   1   1   1   1   2   2   2   2   2   2   2 |..7
↦  -7 :  -   -   -   -   -   -   -   -   1   1   1   1   1   1   1 |..8
↦  -7 :  -   -   -   -   -   -   -   -   1   1   1   1   1   1   1 |..9
↦      === === === === === === === === === === === === === === ===
↦       -7  -7  -7  -7  -7  -7  -7  -7  -6  -6  -6  -6  -6  -6  -6
↦ d_3 :
↦ Graded homomorphism: R(7)^8 + R(6)^7 <- R(6)^3 + R(5)^4 + R(4)^2, given b\
   y a matrix, with degrees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ....
↦       --- --- --- --- --- --- --- --- --- +...
↦   -7 :  1   1   1   2   2   -   -   -   - |..1
↦   -7 :  1   1   1   2   2   -   -   -   - |..2
↦   -7 :  1   1   1   2   2   2   2   3   3 |..3
↦   -7 :  1   1   1   2   2   2   2   3   3 |..4
↦   -7 :  1   1   1   2   2   2   2   3   3 |..5
↦   -7 :  1   1   1   2   2   2   2   3   3 |..6
↦   -7 :  1   1   1   2   2   2   2   3   3 |..7
↦   -7 :  1   1   1   2   2   2   2   3   3 |..8
↦   -6 :  -   -   -   1   1   1   1   2   2 |..9
↦   -6 :  -   -   -   1   1   1   1   2   2 |.10
↦   -6 :  -   -   -   1   1   1   1   2   2 |.11
↦   -6 :  -   -   -   1   1   1   1   2   2 |.12
↦   -6 :  -   -   -   1   1   1   1   2   2 |.13
↦   -6 :  -   -   -   1   1   1   1   2   2 |.14
↦   -6 :  -   -   -   1   1   1   1   2   2 |.15
↦       === === === === === === === === ===
↦        -6  -6  -6  -5  -5  -5  -5  -4  -4
↦ d_4 :
↦ Graded homomorphism: R(6)^3 + R(5)^4 + R(4)^2 <- R, given by a matrix, wi\
   th degrees:
↦       ..1 ....
↦       --- +...
↦   -6 :  6 |..1
↦   -6 :  6 |..2
↦   -6 :  6 |..3
↦   -5 :  5 |..4
↦   -5 :  5 |..5
↦   -5 :  5 |..6
↦   -5 :  5 |..7
↦   -4 :  4 |..8
↦   -4 :  4 |..9
↦       ===
↦         0
↦ 
↦ 
string Name = "rat.d10.g9.quart2"; int @p=31991; ring R = (@p),(x,y,z,u,v), dp;ideal
↦ ================================================
↦ 
↦ === Example: [ rat.d10.g9.quart2 ]
↦  = Ring:  (ZZ/31991),(x,y,z,u,v),(dp(5),C)
↦  ! Resolution via 'grres':
↦  = Non-minimal betti numbers:
↦             0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:     -     -     -     -     -
↦     3:     -     2     -     -     -
↦     4:     -     5     9     3     -
↦     5:     -     1     3     3     1
↦ ------------------------------------
↦ total:     1     8    12     6     1
↦ 
↦  = Degrees of (ordered) maps:
↦ Graded resolution:
↦ R <-- d_1 --
↦ R(-4)^2 + R(-5)^5 + R(-6) <-- d_2 --
↦ R(-6)^9 + R(-7)^3 <-- d_3 --
↦ R(-7)^3 + R(-8)^3 <-- d_4 --
↦ R(-9), given by maps:
↦ d_1 :
↦ Graded homomorphism: R <- R(-4)^2 + R(-5)^5 + R(-6), given by a matrix, w\
  ith degrees:
↦     .1 .2 .3 .4 .5 .6 .7 .8 ...
↦     -- -- -- -- -- -- -- -- +..
↦  0 : 4  4  5  5  5  5  5  6 |.1
↦     == == == == == == == ==
↦      4  4  5  5  5  5  5  6
↦ d_2 :
↦ Graded homomorphism: R(-4)^2 + R(-5)^5 + R(-6) <- R(-6)^9 + R(-7)^3, give\
  n by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   4 :  2   2   2   2   2   2   2   2   2   -   3   3 |..1
↦   4 :  2   2   2   2   2   2   2   2   2   -   3   3 |..2
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..3
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..4
↦   5 :  1   1   1   1   1   1   1   1   1   2   2   2 |..5
↦   5 :  1   1   1   1   1   1   1   1   1   -   2   2 |..6
↦   5 :  1   1   1   1   1   1   1   1   1   -   2   2 |..7
↦   6 :  -   -   -   -   -   -   -   -   -   1   1   1 |..8
↦      === === === === === === === === === === === ===
↦        6   6   6   6   6   6   6   6   6   7   7   7
↦ d_3 :
↦ Graded homomorphism: R(-6)^9 + R(-7)^3 <- R(-7)^3 + R(-8)^3, given by a m\
  atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   6 :  1   1   1   2   2   2 |..1
↦   6 :  1   1   1   2   2   2 |..2
↦   6 :  1   1   1   2   2   2 |..3
↦   6 :  1   1   1   2   2   2 |..4
↦   6 :  1   1   1   2   2   2 |..5
↦   6 :  1   1   1   2   2   2 |..6
↦   6 :  1   1   1   2   2   2 |..7
↦   6 :  1   1   1   2   2   2 |..8
↦   6 :  1   1   1   -   -   2 |..9
↦   7 :  -   -   -   1   1   1 |.10
↦   7 :  -   -   -   1   1   1 |.11
↦   7 :  -   -   -   1   1   1 |.12
↦      === === === === === ===
↦        7   7   7   8   8   8
↦ d_4 :
```

```
↦ Graded homomorphism: R(-7)^3 + R(-8)^3 <- R(-9), given by a matrix, with \
  degrees:
↦     .1 ...
↦     -- +..
↦  7 : 2 |.1
↦  7 : 2 |.2
↦  7 : 2 |.3
↦  8 : 1 |.4
↦  8 : 1 |.5
↦  8 : 1 |.6
↦     ==
↦      9
↦  = TRANSPOSE'd complex: %%%%%%%%%%%%%
↦            0     1     2     3     4
↦ ------------------------------------
↦    -9:     1     3     3     1     -
↦    -8:     -     3     9     5     -
↦    -7:     -     -     -     2     -
↦    -6:     -     -     -     -     -
↦    -5:     -     -     -     -     -
↦    -4:     -     -     -     -     1
↦ ------------------------------------
↦ total:     1     6    12     8     1
↦ 
↦ Graded resolution:
↦ R(9) <-- d_1 --
↦ R(8)^3 + R(7)^3 <-- d_2 --
↦ R(7)^3 + R(6)^9 <-- d_3 --
↦ R(6) + R(5)^5 + R(4)^2 <-- d_4 --
↦ R, given by maps:
↦ d_1 :
↦ Graded homomorphism: R(9) <- R(8)^3 + R(7)^3, given by a matrix, with deg\
  rees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦  -9 :  1   1   1   2   2   2 |..1
↦      === === === === === ===
↦       -8  -8  -8  -7  -7  -7
↦ d_2 :
↦ Graded homomorphism: R(8)^3 + R(7)^3 <- R(7)^3 + R(6)^9, given by a matri\
  x, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- +...
↦  -8 :  1   1   1   2   2   2   2   2   2   2   2   - |..1
↦  -8 :  1   1   1   2   2   2   2   2   2   2   2   - |..2
↦  -8 :  1   1   1   2   2   2   2   2   2   2   2   2 |..3
↦  -7 :  -   -   -   1   1   1   1   1   1   1   1   1 |..4
↦  -7 :  -   -   -   1   1   1   1   1   1   1   1   1 |..5
↦  -7 :  -   -   -   1   1   1   1   1   1   1   1   1 |..6
↦      === === === === === === === === === === === ===
↦       -7  -7  -7  -6  -6  -6  -6  -6  -6  -6  -6  -6
↦ d_3 :
↦ Graded homomorphism: R(7)^3 + R(6)^9 <- R(6) + R(5)^5 + R(4)^2, given by \
  a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ....
↦      --- --- --- --- --- --- --- --- +...
↦   -7 :  1   2   2   2   -   -   -   - |..1
↦   -7 :  1   2   2   2   2   2   3   3 |..2
↦   -7 :  1   2   2   2   2   2   3   3 |..3
↦   -6 :  -   1   1   1   1   1   2   2 |..4
↦   -6 :  -   1   1   1   1   1   2   2 |..5
↦   -6 :  -   1   1   1   1   1   2   2 |..6
↦   -6 :  -   1   1   1   1   1   2   2 |..7
↦   -6 :  -   1   1   1   1   1   2   2 |..8
↦   -6 :  -   1   1   1   1   1   2   2 |..9
↦   -6 :  -   1   1   1   1   1   2   2 |.10
↦   -6 :  -   1   1   1   1   1   2   2 |.11
↦   -6 :  -   1   1   1   1   1   2   2 |.12
↦      === === === === === === === ===
↦       -6  -5  -5  -5  -5  -5  -4  -4
↦  d_4 :
↦  Graded homomorphism: R(6) + R(5)^5 + R(4)^2 <- R, given by a matrix, with\
   degrees:
↦      ..1 ....
↦      --- +...
↦   -6 :  6 |..1
↦   -5 :  5 |..2
↦   -5 :  5 |..3
↦   -5 :  5 |..4
↦   -5 :  5 |..5
↦   -5 :  5 |..6
↦   -4 :  4 |..7
↦   -4 :  4 |..8
↦      ===
↦        0
↦
↦
//  if( defined(assumeLevel0) ){ assumeLevel = assumeLevel0; } else { kill assumeLeve
```

D.15.6.17 KeneshlouMatrixPresentation

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: KeneshlouMatrixPresentation(intvec a), intvec a.

Return: graded object

Purpose: matrix presentation for direct sum of omega^a[i](i) in form of a graded object

Example:

```
LIB "gradedModules.lib";
ring r = 32003,(x(0..4)),dp;
def N1 = KeneshlouMatrixPresentation(intvec(2,0,0,0,0));
grview(N1);
↦  Graded homomorphism: r^2 <- 0, given by zero (2 x 0) matrix.
def N2 = KeneshlouMatrixPresentation(intvec(0,0,0,0,3));
grview(N2);
↦  Graded homomorphism: r(-1)^3 <- 0, given by zero (3 x 0) matrix.
```

```
def N = KeneshlouMatrixPresentation(intvec(2,0,0,0,3));
grview(N);
↦ Graded homomorphism: r^2 + r(-1)^3 <- 0, given by zero (5 x 0) matrix.
def M1 = KeneshlouMatrixPresentation(intvec(0,1,0,0,0));
grview(M1);
↦ Graded homomorphism: r(-1)^10 <- r(-2)^10, given by a square matrix, with\
    degrees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦       --- --- --- --- --- --- --- --- --- --- +...
↦   1 :   1   1   -   -   1   -   -   -   -   - |..1
↦   1 :   1   -   1   -   -   1   -   -   -   - |..2
↦   1 :   1   -   -   1   -   -   1   -   -   - |..3
↦   1 :   -   1   1   -   -   -   -   1   -   - |..4
↦   1 :   -   1   -   1   -   -   -   -   1   - |..5
↦   1 :   -   -   1   1   -   -   -   -   -   1 |..6
↦   1 :   -   -   -   -   1   1   -   1   -   - |..7
↦   1 :   -   -   -   -   1   -   1   -   1   - |..8
↦   1 :   -   -   -   -   -   1   1   -   -   1 |..9
↦   1 :   -   -   -   -   -   -   -   1   1   1 |.10
↦       === === === === === === === === === ===
↦         2   2   2   2   2   2   2   2   2   2
def M2 = KeneshlouMatrixPresentation(intvec(0,1,1,0,0));
grview(M2);
↦ Graded homomorphism: r(-1)^20 <- r(-2)^15, given by a matrix, with degree\
   s:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦       --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   1 :   1   1   -   -   1   -   -   -   -   -   -   -   -   -   - |..1
↦   1 :   1   -   1   -   -   1   -   -   -   -   -   -   -   -   - |..2
↦   1 :   1   -   -   1   -   -   1   -   -   -   -   -   -   -   - |..3
↦   1 :   -   1   1   -   -   -   -   1   -   -   -   -   -   -   - |..4
↦   1 :   -   1   -   1   -   -   -   -   1   -   -   -   -   -   - |..5
↦   1 :   -   -   1   1   -   -   -   -   -   1   -   -   -   -   - |..6
↦   1 :   -   -   -   -   1   1   -   1   -   -   -   -   -   -   - |..7
↦   1 :   -   -   -   -   1   -   1   -   1   -   -   -   -   -   - |..8
↦   1 :   -   -   -   -   -   1   1   -   -   1   -   -   -   -   - |..9
↦   1 :   -   -   -   -   -   -   -   1   1   1   -   -   -   -   - |.10
↦   1 :   -   -   -   -   -   -   -   -   -   -   1   1   -   -   - |.11
↦   1 :   -   -   -   -   -   -   -   -   -   -   1   -   1   -   - |.12
↦   1 :   -   -   -   -   -   -   -   -   -   -   1   -   -   1   - |.13
↦   1 :   -   -   -   -   -   -   -   -   -   -   1   -   -   -   1 |.14
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   1   1   -   - |.15
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   1   -   1   - |.16
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   1   -   -   1 |.17
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   -   1   1   - |.18
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   -   1   -   1 |.19
↦   1 :   -   -   -   -   -   -   -   -   -   -   -   -   -   1   1 |.20
↦       === === === === === === === === === === === === === === ===
↦         2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
def M3 = KeneshlouMatrixPresentation(intvec(0,0,0,1,0));
grview(M3);
↦ Graded homomorphism: r(-1)^5 <- r(-2), given by a matrix, with degrees:
↦      .1 ...
↦     -- +..
↦  1 : 1 |.1
↦  1 : 1 |.2
↦  1 : 1 |.3
↦  1 : 1 |.4
↦  1 : 1 |.5
↦     ==
↦      2
def M = KeneshlouMatrixPresentation(intvec(1,1,1,0,0));
grview(M);
↦ Graded homomorphism: r + r(-1)^20 <- r(-2)^15, given by a matrix, with de\
   grees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   - |..1
↦   1 :  1   1   -   -   1   -   -   -   -   -   -   -   -   -   - |..2
↦   1 :  1   -   1   -   -   1   -   -   -   -   -   -   -   -   - |..3
↦   1 :  1   -   -   1   -   -   1   -   -   -   -   -   -   -   - |..4
↦   1 :  -   1   1   -   -   -   -   1   -   -   -   -   -   -   - |..5
↦   1 :  -   1   -   1   -   -   -   -   1   -   -   -   -   -   - |..6
↦   1 :  -   -   1   1   -   -   -   -   -   1   -   -   -   -   - |..7
↦   1 :  -   -   -   -   1   1   -   1   -   -   -   -   -   -   - |..8
↦   1 :  -   -   -   -   1   -   1   -   1   -   -   -   -   -   - |..9
↦   1 :  -   -   -   -   -   1   1   -   -   1   -   -   -   -   - |.10
↦   1 :  -   -   -   -   -   -   -   1   1   1   -   -   -   -   - |.11
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   1   -   -   - |.12
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   1   -   - |.13
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   -   1   - |.14
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   -   -   1 |.15
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   1   -   - |.16
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   -   1   - |.17
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   -   -   1 |.18
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   1   1   - |.19
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   1   -   1 |.20
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   1   1 |.21
↦      === === === === === === === === === === === === === === ===
↦        2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

D.15.6.18 grsyz

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grsyz(M), graded object M

Return: graded object

Purpose: compute graded syzygy of M

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grgroebner( grobj( module([x+y, x, 0, 3], [0, x+y, y, 2], [y, y, z, 1]),
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3 + r(-2) + r(-3), given by a m\
   atrix, with degrees:
```

```
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   1   1   2   - |..1
↦   0 :  1   -   1   -   - |..2
↦   0 :  1   1   1   2   3 |..3
↦   1 :  0   0   0   1   2 |..4
↦      === === === === ===
↦        1   1   1   2   3
grview(grsyz(A));
↦ Graded homomorphism: r(-1)^3 + r(-2) + r(-3) <- r(-2) + r(-3), given by a\
    matrix, with degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   1 :  1   - |..1
↦   1 :  1   2 |..2
↦   1 :  1   - |..3
↦   2 :  0   1 |..4
↦   3 :  -   0 |..5
↦      === ===
↦        2   3
module X = grgroebner( grobj( module([x]), intvec(2) ) );
grview(X);
↦ Graded homomorphism: r(-2) <- r(-3), given by a diagonal matrix, with deg\
   rees:
↦     .1 ...
↦     -- +..
↦  2 : 1 |.1
↦     ==
↦      3
// syzygy module should be zero!
grview(grsyz(X));
↦ Graded homomorphism: r(-3) <- 0, given by zero (1 x 0) matrix.
```

D.15.6.19 grres

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grres(M, l[, b]), graded object M, int l, int b

Return: graded resolution = list of graded objects

Purpose: compute graded resolution of M (of length l) and minimise it if b was given

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 3], [0, x+y, y, 2], [y, y, z, 1]), intvec(0,0,0
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3, given by a matrix, with degr\
   ees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :  1   -   1 |..1
↦   0 :  1   1   1 |..2
↦   0 :  -   1   1 |..3
↦   1 :  0   0   0 |..4
↦      === === ===
↦        1   1   1
module B = grgroebner(A);
grview(B);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3 + r(-2) + r(-3), given by a m\
  atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   1   1   2   - |..1
↦   0 :  1   -   1   -   - |..2
↦   0 :  1   1   1   2   3 |..3
↦   1 :  0   0   0   1   2 |..4
↦      === === === === ===
↦        1   1   1   2   3
"graded resolution of B: "; def C = grres(B, 0); grview(C);
↦ graded resolution of B:
↦ Graded resolution:
↦ r^3 + r(-1) <-- d_1 --
↦ r(-1) + r(-2) + r(-1)^2 + r(-3) <-- d_2 --
↦ r(-3) + r(-2) <-- d_3 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: r^3 + r(-1) <- r(-1) + r(-2) + r(-1)^2 + r(-3), give\
  n by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   2   1   1   - |..1
↦   0 :  -   -   1   1   - |..2
↦   0 :  1   2   1   1   3 |..3
↦   1 :  0   1   0   0   2 |..4
↦      === === === === ===
↦        1   2   1   1   3
↦ d_2 :
↦ Graded homomorphism: r(-1) + r(-2) + r(-1)^2 + r(-3) <- r(-3) + r(-2), gi\
  ven by a matrix, with degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   1 :  2   1 |..1
↦   2 :  1   0 |..2
↦   1 :  -   1 |..3
↦   1 :  -   1 |..4
↦   3 :  0   - |..5
↦      === ===
↦        3   2
↦ d_3 :
↦ Graded homomorphism: r(-3) + r(-2) <- 0, given by zero (2 x 0) matrix.
int i; int l = size(C);
"D^2 == 0: "; for (i = 1; i < l; i++ ) { i; grview( grprod(C[i], C[i+1]) ); }
↦ D^2 == 0:
↦ 1
↦ Graded homomorphism: r^3 + r(-1) <- r(-3) + r(-2), given by zero (4 x 2) \
  matrix.
```

```
↦ 2
↦ Graded homomorphism: r(-1) + r(-2) + r(-1)^2 + r(-3) <- 0, given by zero \
   (5 x 0) matrix.
```

D.15.6.20 grlift

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grlift(M, N), graded objects M and N

Return: transformation matrix (graded object???)

Purpose: compute graded matrix which the generators of submodule Im(N) in terms of Im(M).

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module P=grobj(module([xy,0,xz]),intvec(0,1,0));
grview(P);
↦ Graded homomorphism: r + r(-1) + r <- r(-2), given by a matrix, with degr\
   ees:
↦       ..1 ....
↦       --- +...
↦    0 :  2 |..1
↦    1 :  - |..2
↦    0 :  2 |..3
↦       ===
↦         2
module D=grobj(module([y,0,z],[x2+y2,z,0]),intvec(0,1,0));
grview(D);
↦ Graded homomorphism: r + r(-1) + r <- r(-1) + r(-2), given by a matrix, w\
   ith degrees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   2 |..1
↦    1 :  -   1 |..2
↦    0 :  1   - |..3
↦       === ===
↦         1   2
def G=grlift(D,P);
grview(G);
↦ Graded homomorphism: r(-1) + r(-2) <- r(-2), given by a matrix, with degr\
   ees:
↦       ..1 ....
↦       --- +...
↦    1 :  1 |..1
↦    2 :  - |..2
↦       ===
↦         2
ASSUME(0, grisequal( grprod(D, G), P) );
```

D.15.6.21 grprod

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grprod(M, N), graded objects M and N

Return: graded object

Purpose: compute graded product M * N (as composition of maps)

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 3], [0, x+y, y, 2], [y, y, z, 1]), intvec(0,0,0
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3, given by a matrix, with degr\
   ees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :  1   -   1 |..1
↦   0 :  1   1   1 |..2
↦   0 :  -   1   1 |..3
↦   1 :  0   0   0 |..4
↦      === === ===
↦        1   1   1
A = grgroebner(A);
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^3 + r(-2) + r(-3), given by a m\
   atrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ....
↦      --- --- --- --- --- +...
↦   0 :  1   1   1   2   - |..1
↦   0 :  1   -   1   -   - |..2
↦   0 :  1   1   1   2   3 |..3
↦   1 :  0   0   0   1   2 |..4
↦      === === === === ===
↦        1   1   1   2   3
module B = grsyz(A);
grview(B);
↦ Graded homomorphism: r(-1)^3 + r(-2) + r(-3) <- r(-2) + r(-3), given by a\
    matrix, with degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   1 :  1   - |..1
↦   1 :  1   2 |..2
↦   1 :  1   - |..3
↦   2 :  0   1 |..4
↦   3 :  -   0 |..5
↦      === ===
↦        2   3
print(B);
↦ x, 0,
↦ -y,y2,
↦ -y,0,
↦ 1, -x-2y,
↦ 0, 1
module D = grprod( A, B );
grview(D);
↦ Graded homomorphism: r^3 + r(-1) <- r(-2) + r(-3), given by zero (4 x 2) \
   matrix.
```

```
print(D); // must be all zeroes due to syzygy property!
↦ 0,0,
↦ 0,0,
↦ 0,0,
↦ 0,0
ASSUME(0, size(D) == 0);
```

D.15.6.22 grgroebner

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grgroebner(M), graded object M

Return: graded object

Purpose: compute graded groebner basis of M

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module A = grobj( module([x+y, x, 0, 0], [0, x+y, y, 0]), intvec(0,0,0,1) );
grview(A);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
   ees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   - |..1
↦    0 :  1   1 |..2
↦    0 :  -   1 |..3
↦    1 :  -   - |..4
↦       === ===
↦         1   1
module B = grgroebner(A);
grview(B);
↦ Graded homomorphism: r^3 + r(-1) <- r(-1)^2, given by a matrix, with degr\
   ees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   - |..1
↦    0 :  1   1 |..2
↦    0 :  1   1 |..3
↦    1 :  -   - |..4
↦       === ===
↦         1   1
```

D.15.6.23 grconcat

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grconcat(A, B), graded objects A and B, dst(A) == dst(B) =: dst

Return: graded object

Purpose: construct src(A) + src(B) ——> dst given by (A|B)

Example:

```
LIB "gradedModules.lib";
ring r;
module R=grobj(module([x,y,z]),intvec(0:3));
grview(R);
↦ Graded homomorphism: r^3 <- r(-1), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  0 : 1 |.1
↦  0 : 1 |.2
↦  0 : 1 |.3
↦     ==
↦      1
module S=grobj(module([x,0,y],[xy,zy+x2,0]),intvec(0:3));
grview(S);
↦ Graded homomorphism: r^3 <- r(-1) + r(-2), given by a matrix, with degree\
  s:
↦      ..1 ..2 ....
↦      --- --- +...
↦   0 :  1   2 |..1
↦   0 :  -   2 |..2
↦   0 :  1   - |..3
↦      === ===
↦        1   2
def Q=grconcat(R,S);
grview(Q);
↦ Graded homomorphism: r^3 <- r(-1)^2 + r(-2), given by a square matrix, wi\
  th degrees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   0 :  1   1   2 |..1
↦   0 :  1   -   2 |..2
↦   0 :  1   1   - |..3
↦      === === ===
↦        1   1   2
```

D.15.6.24 grrndmat

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grrndmat(src,dst[,p,b]), intvec src, dst[, int p, b]

Return: matrix of polynomials

Purpose: generate random matrix compatible with src and dst gradings

Note: optional arguments p, b are for 'sparsepoly' (by default: 75%, 30000).

Todo: this is experimental at the moment!

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
print( grrndmat( intvec(0, 1), intvec(1, 2, 3) ) );
↦ 6204y,                    -7833,
↦ -12498xz+12773z2,         5664z,
↦ -10041x3-11973x2y-5644xy2,-7740xy-5361z2
```

D.15.6.25 grrndmap

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grrndmap(S,D), graded objects S and D

Return: graded object

Purpose: construct a random 0-deg graded homomorphism src(S) -> src(D)

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module D=grobj(module([y,0,z],[x2+y2,z,0]),intvec(0,1,0));
grview(D);
↦ Graded homomorphism: r + r(-1) + r <- r(-1) + r(-2), given by a matrix, w\
   ith degrees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   2 |..1
↦    1 :  -   1 |..2
↦    0 :  1   - |..3
↦       === ===
↦         1   2
module S=grobj(module([x,0,y],[xy,zy+x2,0]),intvec(0,0,0));
grview(S);
↦ Graded homomorphism: r^3 <- r(-1) + r(-2), given by a matrix, with degree\
   s:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   2 |..1
↦    0 :  -   2 |..2
↦    0 :  1   - |..3
↦       === ===
↦         1   2
def H=grrndmap(D,S);
grview(H);
↦ Graded homomorphism: r(-1) + r(-2) <- r(-1) + r(-2), given by a square ma\
   trix, with degrees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    1 :  0   1 |..1
↦    2 :  -   0 |..2
↦       === ===
↦         1   2
```

D.15.6.26 grrndmap2

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grrndmap2(D,S), graded objects S and D

Return: graded object

Purpose: construct a random 0-deg graded homomorphism between target of D and S.

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module D=grobj(module([y,0,z],[x2+y2,z,0]),intvec(0,1,0));
grview(D);
↦ Graded homomorphism: r + r(-1) + r <- r(-1) + r(-2), given by a matrix, w\
   ith degrees:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   2 |..1
↦    1 :  -   1 |..2
↦    0 :  1   - |..3
↦       === ===
↦         1   2
module S=grobj(module([x,0,y],[xy,zy+x2,0]),intvec(0,0,0));
grview(S);
↦ Graded homomorphism: r^3 <- r(-1) + r(-2), given by a matrix, with degree\
   s:
↦       ..1 ..2 ....
↦       --- --- +...
↦    0 :  1   2 |..1
↦    0 :  -   2 |..2
↦    0 :  1   - |..3
↦       === ===
↦         1   2
def G=grrndmap2(D,S);
grview(G);
↦ Graded homomorphism: r^3 <- r + r(-1) + r, given by a square matrix, with\
    degrees:
↦      .1 .2 .3 ...
↦      -- -- -- +..
↦   0 : 0  1  0 |.1
↦   0 : 0  1  0 |.2
↦   0 : 0  1  0 |.3
↦      == == ==
↦       0  1  0
```

D.15.6.27 grlifting

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grlifting(M,N), graded objects M and N

Return: map of chain complexes (as a list)

Purpose: construct a map of chain complexes between free resolutions of Img(M) and Img(N).

Example:

```
LIB "gradedModules.lib";
/*
ring r=32003,(x,y,z),dp;
module P=grobj(module([xy,0,xz]),intvec(0,1,0));
grview(P);
module D=grobj(module([y,0,z],[x2+y2,z,0]),intvec(0,1,0));
grview(D);
def G=grlifting(D,P);
```

```
grview(G);
kill r;
ring r=32003,(x,y,z),dp;
module D=grobj(module([y,0,z],[x2+y2,z,0], [z3, xy, xy2]),intvec(0,1,0));
D = grgroebner(D);
grview( grres(D, 0));
def G=grlifting(D, D);
grview(G);
*/
ring S = 0, (x(0..3)), dp;
list kos = grres(grobj(maxideal(1), intvec(0)), 0);
print( betti(kos), "betti");
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     4     6     4     1
↦ ------------------------------------
↦ total:     1     4     6     4     1
↦ 
grview(kos);
↦ Graded resolution:
↦ S <-- d_1 --
↦ S(-1)^4 <-- d_2 --
↦ S(-2)^6 <-- d_3 --
↦ S(-3)^4 <-- d_4 --
↦ S(-4) <-- d_5 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: S <- S(-1)^4, given by a matrix, with degrees:
↦     .1 .2 .3 .4 ...
↦     -- -- -- -- +..
↦  0 : 1  1  1  1 |.1
↦     == == == ==
↦      1  1  1  1
↦ d_2 :
↦ Graded homomorphism: S(-1)^4 <- S(-2)^6, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   1 :   1   1   -   1   -   - |..1
↦   1 :   1   -   1   -   1   - |..2
↦   1 :   -   1   1   -   -   1 |..3
↦   1 :   -   -   -   1   1   1 |..4
↦      === === === === === ===
↦        2   2   2   2   2   2
↦ d_3 :
↦ Graded homomorphism: S(-2)^6 <- S(-3)^4, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦   2 :   1   1   -   - |..1
↦   2 :   1   -   1   - |..2
↦   2 :   1   -   -   1 |..3
↦   2 :   -   1   1   - |..4
↦   2 :   -   1   -   1 |..5
↦   2 :   -   -   1   1 |..6
↦      === === === ===
↦        3   3   3   3
↦ d_4 :
↦ Graded homomorphism: S(-3)^4 <- S(-4), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  3 : 1 |.1
↦  3 : 1 |.2
↦  3 : 1 |.3
↦  3 : 1 |.4
↦     ==
↦      4
↦ d_5 :
↦ Graded homomorphism: S(-4) <- 0, given by zero (1 x 0) matrix.
//  module M = grshift(kos[4], 2); // phi, Syz_3(K(2))
def M = KeneshlouMatrixPresentation(intvec(0,0,1,0));
grview( grres(M, 0) );
↦ Graded resolution:
↦ S(-1)^4 <-- d_1 --
↦ S(-2) <-- d_2 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: S(-1)^4 <- S(-2), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  1 : 1 |.1
↦  1 : 1 |.2
↦  1 : 1 |.3
↦  1 : 1 |.4
↦     ==
↦      2
↦ d_2 :
↦ Graded homomorphism: S(-2) <- 0, given by zero (1 x 0) matrix.
//   module N = grshift(kos[3], 1); // psi, Syz_2(K(1))
def N = KeneshlouMatrixPresentation(intvec(0,1,0,0));
grview( grres(N, 0) );
↦ Graded resolution:
↦ S(-1)^6 <-- d_1 --
↦ S(-2)^4 <-- d_2 --
↦ S(-3) <-- d_3 --
↦ 0, given by maps:
↦ d_1 :
↦ Graded homomorphism: S(-1)^6 <- S(-2)^4, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ....
↦      --- --- --- --- +...
↦   1 :  1   1   -   - |..1
↦   1 :  1   -   1   - |..2
↦   1 :  1   -   -   1 |..3
↦   1 :  -   1   1   - |..4
↦   1 :  -   1   -   1 |..5
↦   1 :  -   -   1   1 |..6
↦      === === === ===
↦        2   2   2   2
↦ d_2 :
↦ Graded homomorphism: S(-2)^4 <- S(-3), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  2 : 1 |.1
↦  2 : 1 |.2
↦  2 : 1 |.3
↦  2 : 1 |.4
↦     ==
↦      3
↦ d_3 :
↦ Graded homomorphism: S(-3) <- 0, given by zero (1 x 0) matrix.
grlifting(M, N); // grview(G);
↦ t:  2
↦ _[1]=26642*gen(4)+24263*gen(3)+5664*gen(2)+24170*gen(1)
//  def G=grlifting( grgens(M), grgens(N) );  grview(G);
```

D.15.6.28 grlifting2

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grlifting2(A,B), graded objects A and B (matrices defining maps)

Return: map of chain complexes (as a list)

Purpose: construct a map of chain complexes between free resolution of M=coker(A) and N=coker(B).

Example:

```
LIB "gradedModules.lib";
ring r;
module P=grobj(module([xy,0,xz]),intvec(0,1,0));
grview(P);
↦ Graded homomorphism: r + r(-1) + r <- r(-2), given by a matrix, with degr\
  ees:
↦      ..1 ....
↦      --- +...
↦   0 :  2 |..1
↦   1 :  - |..2
↦   0 :  2 |..3
↦      ===
↦        2
module D=grobj(module([y,0,z],[x2+y2,z,0]),intvec(0,1,0));
grview(D);
↦ Graded homomorphism: r + r(-1) + r <- r(-1) + r(-2), given by a matrix, w\
  ith degrees:
↦      ..1 ..2 ....
↦      --- --- +...
↦   0 :  1   2 |..1
↦   1 :  -   1 |..2
↦   0 :  1   - |..3
↦      === ===
↦        1   2
module PP = grpres(P);
grview(PP);
```

```
↦ Graded homomorphism: r(-2) <- 0, given by zero (1 x 0) matrix.
module DD = grpres(D);
grview(DD);
↦ Graded homomorphism: r(-1) + r(-2) <- 0, given by zero (2 x 0) matrix.
def T=grlifting2(DD,PP); T;
↦ T[1]=0
↦ T[2]=-5361*gen(1)
// def Z=grlifting2(P,D); Z; // WRONG!!!
```

D.15.6.29 mappingcone

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: mappingcone(M,N), M,N graded objects

Return: chain complex (as a list)

Purpose: construct a free resolution of the cokernel of a random map between Img(M), and Img(N).

Example:

```
LIB "gradedModules.lib";
ring r=32003, (x(0..4)),dp;
def A=KeneshlouMatrixPresentation(intvec(0,0,0,0,3));
def M=grgens(A);
grview(M);
↦ Graded homomorphism: r(-1)^3 <- r(-1)^3, given by a diagonal matrix, with\
    degrees:
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   1 :  0   -   - |..1
↦   1 :  -   0   - |..2
↦   1 :  -   -   0 |..3
↦      === === ===
↦        1   1   1
def B=KeneshlouMatrixPresentation(intvec(0,1,0,0,0));
def N=grgens(B);
grview(N);
↦ Graded homomorphism: r^5 <- r(-1)^10, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  -   -   -   -   -   -   1   1   1   1 |..1
↦   0 :  1   1   -   1   -   -   1   -   -   - |..2
↦   0 :  1   -   1   -   1   -   -   1   -   - |..3
↦   0 :  -   1   1   -   -   1   -   -   1   - |..4
↦   0 :  -   -   -   1   1   1   -   -   -   1 |..5
↦      === === === === === === === === === ===
↦        1   1   1   1   1   1   1   1   1   1
def R=grlifting(M,N);
↦ t:  2
grview(R);
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3, given by a matrix, with degrees\
   :
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   1 :  0   0   0 |..1
↦   1 :  0   0   0 |..2
↦   1 :  0   0   0 |..3
↦   1 :  0   0   0 |..4
↦   1 :  0   0   0 |..5
↦   1 :  0   0   0 |..6
↦   1 :  0   0   0 |..7
↦   1 :  0   0   0 |..8
↦   1 :  0   0   0 |..9
↦   1 :  0   0   0 |.10
↦      === === ===
↦        1   1   1
def T=mappingcone(M,N);
↦ t:  2
grview(T);
↦ Graded resolution:
↦ r(-1)^10 <-- d_1 --
↦ r(-1)^3 + r(-2)^10, given by maps:
↦ d_1 :
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3 + r(-2)^10, given by a matrix, w\
   ith degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   1 :  0   0   0   1   1   -   1   -   -   -   -   -   - |..1
↦   1 :  0   0   0   1   -   1   -   1   -   -   -   -   - |..2
↦   1 :  0   0   0   -   1   1   -   -   1   -   -   -   - |..3
↦   1 :  0   0   0   -   -   -   1   1   1   -   -   -   - |..4
↦   1 :  0   0   0   1   -   -   -   -   -   1   1   -   - |..5
↦   1 :  0   0   0   -   1   -   -   -   -   1   -   1   - |..6
↦   1 :  0   0   0   -   -   -   1   -   -   -   1   1   - |..7
↦   1 :  0   0   0   -   -   1   -   -   -   1   -   -   1 |..8
↦   1 :  0   0   0   -   -   -   -   1   -   -   1   -   1 |..9
↦   1 :  0   0   0   -   -   -   -   -   1   -   -   1   1 |.10
↦      === === === === === === === === === === === === ===
↦        1   1   1   2   2   2   2   2   2   2   2   2   2
def U=grtranspose(T[1]);
resolution G=mres(U,0);
print(betti(G),"betti");
↦            0     1     2
↦ ------------------------
↦    -2:    10     7     -
↦    -1:     -     -     -
↦     0:     -     -     1
↦ ------------------------
↦ total:    10     7     1
↦ 
ideal I=groebner(flatten(G[2]));
resolution GI=mres(I,0);
print(betti(GI),"betti");
↦            0     1     2     3     4
↦ ------------------------------------
↦     0:     1     -     -     -     -
↦     1:     -     -     -     -     -
↦     2:      -      7     10      5      1
↦ ------------------------------------
↦ total:      1      7     10      5      1
↦
```

D.15.6.30 grlifting3

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Todo: grlifting4 was newer and had more documentation than this proc, but was removed... Please verify and update!

Example:

```
LIB "gradedModules.lib";
ring r=32003, x(0..4),dp;
def A=grtwist(3,1);
grview(A);
↦ Graded homomorphism: r(1)^3 <- 0, given by zero (3 x 0) matrix.
def T=KeneshlouMatrixPresentation(intvec(0,1,0,0,0));
grview(T);
↦ Graded homomorphism: r(-1)^10 <- r(-2)^10, given by a square matrix, with\
   degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   1 :   1   1   -   -   1   -   -   -   -   - |..1
↦   1 :   1   -   1   -   -   1   -   -   -   - |..2
↦   1 :   1   -   -   1   -   -   1   -   -   - |..3
↦   1 :   -   1   1   -   -   -   -   1   -   - |..4
↦   1 :   -   1   -   1   -   -   -   -   1   - |..5
↦   1 :   -   -   1   1   -   -   -   -   -   1 |..6
↦   1 :   -   -   -   -   1   1   -   1   -   - |..7
↦   1 :   -   -   -   -   1   -   1   -   1   - |..8
↦   1 :   -   -   -   -   -   1   1   -   -   1 |..9
↦   1 :   -   -   -   -   -   -   -   1   1   1 |.10
↦      === === === === === === === === === ===
↦        2   2   2   2   2   2   2   2   2   2
def F=grlifting3(T,A);
↦            0      1      2      3
↦ ------------------------------
↦     1:     10     10      5      1
↦ ------------------------------
↦ total:     10     10      5      1
↦
↦            0
↦ ------------
↦    -1:      3
↦ ------------
↦ total:      3
↦
↦ t:  1
↦ Graded homomorphism: r(1)^3 <- r(-1)^10, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦  -1 :   2   2   2   2   2   2   2   2   2   2 |..1
↦  -1 :  2   2   2   2   2   2   2   2   2   2 |..2
↦  -1 :  2   2   2   2   2   2   2   2   2   2 |..3
↦      === === === === === === === === === ===
↦        1   1   1   1   1   1   1   1   1   1
grview(F);
↦ Graded resolution:
↦ r(1)^3 <-- d_1 --
↦ r(-1)^10, given by maps:
↦ d_1 :
↦ Graded homomorphism: r(1)^3 <- r(-1)^10, given by a matrix, with degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦  -1 :  2   2   2   2   2   2   2   2   2   2 |..1
↦  -1 :  2   2   2   2   2   2   2   2   2   2 |..2
↦  -1 :  2   2   2   2   2   2   2   2   2   2 |..3
↦      === === === === === === === === === ===
↦        1   1   1   1   1   1   1   1   1   1
def R=KeneshlouMatrixPresentation(intvec(0,0,0,2,0));
def S=KeneshlouMatrixPresentation(intvec(1,2,0,0,0));
def H=grlifting3(R, S);
↦            0     1
↦ ------------------
↦     1:    10     2
↦ ------------------
↦ total:    10     2
↦ 
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:    20    20    10     2
↦ ------------------------------
↦ total:    21    20    10     2
↦ 
↦ t:  2
↦ Graded homomorphism: r(-2)^20 <- r(-2)^2, given by a matrix, with degrees\
  :
↦      ..1 ..2 ....
↦      --- --- +...
↦   2 :  0   0 |..1
↦   2 :  0   0 |..2
↦   2 :  0   0 |..3
↦   2 :  0   0 |..4
↦   2 :  0   0 |..5
↦   2 :  0   0 |..6
↦   2 :  0   0 |..7
↦   2 :  0   0 |..8
↦   2 :  0   0 |..9
↦   2 :  0   0 |.10
↦   2 :  0   0 |.11
↦   2 :  0   0 |.12
↦   2 :  0   0 |.13
↦   2 :  0   0 |.14
↦   2 :  0   0 |.15
↦   2 :  0   0 |.16
↦   2 :  0   0 |.17
↦   2 :  0   0 |.18
↦   2 :  0   0 |.19
↦   2 :  0   0 |.20
↦      === ===
↦        2   2
↦ k:  1
↦ Graded homomorphism: r + r(-1)^20 <- r(-1)^10, given by a matrix, with de\
  grees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  -   -   -   -   -   -   -   -   -   - |..1
↦   1 :  0   0   0   -   -   0   0   0   -   - |..2
↦   1 :  0   0   -   0   -   0   0   -   0   - |..3
↦   1 :  0   0   -   -   0   0   0   -   -   0 |..4
↦   1 :  0   -   0   0   -   0   -   0   0   - |..5
↦   1 :  0   -   0   -   0   0   -   0   -   0 |..6
↦   1 :  0   -   -   0   0   0   -   -   0   0 |..7
↦   1 :  -   0   0   0   -   -   0   0   0   - |..8
↦   1 :  -   0   0   -   0   -   0   0   -   0 |..9
↦   1 :  -   0   -   0   0   -   0   -   0   0 |.10
↦   1 :  -   -   0   0   0   -   -   0   0   0 |.11
↦   1 :  0   0   0   -   -   0   0   0   -   - |.12
↦   1 :  0   0   -   0   -   0   0   -   0   - |.13
↦   1 :  0   0   -   -   0   0   0   -   -   0 |.14
↦   1 :  0   -   0   0   -   0   -   0   0   - |.15
↦   1 :  0   -   0   -   0   0   -   0   -   0 |.16
↦   1 :  0   -   -   0   0   0   -   -   0   0 |.17
↦   1 :  -   0   0   0   -   -   0   0   0   - |.18
↦   1 :  -   0   0   -   0   -   0   0   -   0 |.19
↦   1 :  -   0   -   0   0   -   0   -   0   0 |.20
↦   1 :  -   -   0   0   0   -   -   0   0   0 |.21
↦      === === === === === === === === === ===
↦        1   1   1   1   1   1   1   1   1   1
// grview(H);
// 2nd module does not lie in the first:
// def H=grlifting3(S, R);
//def I=KeneshlouMatrixPresentation(intvec(2,3,0,6,2));
//def J=KeneshlouMatrixPresentation(intvec(4,0,1,2,1));
//def N=grlifting3(I,J); grview(N);
```

D.15.6.31 mappingcone3

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: mappingcone3(A,B), graded objects A and B (matrices defining maps)

Return: chain complex (as a list)

Purpose: construct a free resolution of the cokernel of a random map between M=coker(A), and N=coker(B)

Example:

```
LIB "gradedModules.lib";
```

```
ring r=32003,x(0..4),dp;
def A=KeneshlouMatrixPresentation(intvec(0,0,0,0,3));
grview(A);
↦ Graded homomorphism: r(-1)^3 <- 0, given by zero (3 x 0) matrix.
def T= KeneshlouMatrixPresentation(intvec(0,1,0,0,0));
grview(T);
↦ Graded homomorphism: r(-1)^10 <- r(-2)^10, given by a square matrix, with\
   degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   1 :  1   1   -   -   1   -   -   -   -   - |..1
↦   1 :  1   -   1   -   -   1   -   -   -   - |..2
↦   1 :  1   -   -   1   -   -   1   -   -   - |..3
↦   1 :  -   1   1   -   -   -   -   1   -   - |..4
↦   1 :  -   1   -   1   -   -   -   -   1   - |..5
↦   1 :  -   -   1   1   -   -   -   -   -   1 |..6
↦   1 :  -   -   -   -   1   1   -   1   -   - |..7
↦   1 :  -   -   -   -   1   -   1   -   1   - |..8
↦   1 :  -   -   -   -   -   1   1   -   -   1 |..9
↦   1 :  -   -   -   -   -   -   -   1   1   1 |.10
↦      === === === === === === === === === ===
↦        2   2   2   2   2   2   2   2   2   2
def F=grlifting3(A,T); grview(F);
↦            0
↦ ------------
↦     1:     3
↦ ------------
↦ total:     3
↦ 
↦            0     1     2     3
↦ ------------------------------
↦     1:    10    10     5     1
↦ ------------------------------
↦ total:    10    10     5     1
↦ 
↦ t:  1
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3, given by a matrix, with degrees\
  :
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   1 :  0   0   0 |..1
↦   1 :  0   0   0 |..2
↦   1 :  0   0   0 |..3
↦   1 :  0   0   0 |..4
↦   1 :  0   0   0 |..5
↦   1 :  0   0   0 |..6
↦   1 :  0   0   0 |..7
↦   1 :  0   0   0 |..8
↦   1 :  0   0   0 |..9
↦   1 :  0   0   0 |.10
↦      === === ===
↦        1   1   1
↦ Graded resolution:
```

```
↦ r(-1)^10 <-- d_1 --
↦ r(-1)^3, given by maps:
↦ d_1 :
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3, given by a matrix, with degrees\
  :
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   1 :  0   0   0 |..1
↦   1 :  0   0   0 |..2
↦   1 :  0   0   0 |..3
↦   1 :  0   0   0 |..4
↦   1 :  0   0   0 |..5
↦   1 :  0   0   0 |..6
↦   1 :  0   0   0 |..7
↦   1 :  0   0   0 |..8
↦   1 :  0   0   0 |..9
↦   1 :  0   0   0 |.10
↦      === === ===
↦        1   1   1
// BUG in the proc
def G=mappingcone3(A,T); grview(G);
↦            0
↦ ------------
↦     1:     3
↦ ------------
↦ total:     3
↦
↦            0     1     2     3
↦ ------------------------------
↦     1:    10    10     5     1
↦ ------------------------------
↦ total:    10    10     5     1
↦
↦ t:  1
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3, given by a matrix, with degrees\
  :
↦      ..1 ..2 ..3 ....
↦      --- --- --- +...
↦   1 :  0   0   0 |..1
↦   1 :  0   0   0 |..2
↦   1 :  0   0   0 |..3
↦   1 :  0   0   0 |..4
↦   1 :  0   0   0 |..5
↦   1 :  0   0   0 |..6
↦   1 :  0   0   0 |..7
↦   1 :  0   0   0 |..8
↦   1 :  0   0   0 |..9
↦   1 :  0   0   0 |.10
↦      === === ===
↦        1   1   1
↦ Graded resolution:
↦ r(-1)^10 <-- d_1 --
↦ r(-1)^3 + r(-2)^10, given by maps:
```

```
↦ d_1 :
↦ Graded homomorphism: r(-1)^10 <- r(-1)^3 + r(-2)^10, given by a matrix, w\
   ith degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦   1 :  0   0   0   1   1   -   -   1   -   -   -   -   - |..1
↦   1 :  0   0   0   1   -   1   -   -   1   -   -   -   - |..2
↦   1 :  0   0   0   1   -   -   1   -   -   1   -   -   - |..3
↦   1 :  0   0   0   -   1   1   -   -   -   -   1   -   - |..4
↦   1 :  0   0   0   -   1   -   1   -   -   -   -   1   - |..5
↦   1 :  0   0   0   -   -   1   1   -   -   -   -   -   1 |..6
↦   1 :  0   0   0   -   -   -   -   1   1   -   1   -   - |..7
↦   1 :  0   0   0   -   -   -   -   1   -   1   -   1   - |..8
↦   1 :  0   0   0   -   -   -   -   -   1   1   -   -   1 |..9
↦   1 :  0   0   0   -   -   -   -   -   -   -   1   1   1 |.10
↦      === === === === === === === === === === === === ===
↦        1   1   1   2   2   2   2   2   2   2   2   2   2
/*
module W=grtranspose(G[1]);
resolution U=mres(W,0);
print(betti(U,0),"betti"); // ?
ideal P=groebner(flatten(U[2]));
resolution L=mres(P,0);
print(betti(L),"betti");
*/
def R=KeneshlouMatrixPresentation(intvec(0,0,0,2,0));
grview(R);
↦ Graded homomorphism: r(-1)^10 <- r(-2)^2, given by a matrix, with degrees\
   :
↦      ..1 ..2 ....
↦      --- --- +...
↦   1 :  1   - |..1
↦   1 :  1   - |..2
↦   1 :  1   - |..3
↦   1 :  1   - |..4
↦   1 :  1   - |..5
↦   1 :  -   1 |..6
↦   1 :  -   1 |..7
↦   1 :  -   1 |..8
↦   1 :  -   1 |..9
↦   1 :  -   1 |.10
↦      === ===
↦        2   2
def S=KeneshlouMatrixPresentation(intvec(1,2,0,0,0));
grview(S);
↦ Graded homomorphism: r + r(-1)^20 <- r(-2)^20, given by a matrix, with de\
   grees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 .16 .17 \
   .18 .19 .20 ....
↦      --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- -
↦   0 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   - \
    -   -   - |..1
↦   1 :  1   1   -   -   1   -   -   -   -   -   -   -   -   -   -   -   - \
     -   -   - |..2
↦    1 :  1   -   1   -   -   1   -   -   -   -   -   -   -   -   -   -   - \
     -   -   - |..3
↦    1 :  1   -   -   1   -   -   1   -   -   -   -   -   -   -   -   -   - \
     -   -   - |..4
↦    1 :  -   1   1   -   -   -   -   1   -   -   -   -   -   -   -   -   - \
     -   -   - |..5
↦    1 :  -   1   -   1   -   -   -   -   1   -   -   -   -   -   -   -   - \
     -   -   - |..6
↦    1 :  -   -   1   1   -   -   -   -   -   1   -   -   -   -   -   -   - \
     -   -   - |..7
↦    1 :  -   -   -   -   1   1   -   1   -   -   -   -   -   -   -   -   - \
     -   -   - |..8
↦    1 :  -   -   -   -   1   -   1   -   1   -   -   -   -   -   -   -   - \
     -   -   - |..9
↦    1 :  -   -   -   -   -   1   1   -   -   1   -   -   -   -   -   -   - \
     -   -   - |.10
↦    1 :  -   -   -   -   -   -   -   1   1   1   -   -   -   -   -   -   - \
     -   -   - |.11
↦    1 :  -   -   -   -   -   -   -   -   -   -   1   1   -   -   1   -   - \
     -   -   - |.12
↦    1 :  -   -   -   -   -   -   -   -   -   -   1   -   1   -   -   1   - \
     -   -   - |.13
↦    1 :  -   -   -   -   -   -   -   -   -   -   1   -   -   1   -   -   1 \
     -   -   - |.14
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   1   1   -   -   -   - \
     1   -   - |.15
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   1   -   1   -   -   - \
     -   1   - |.16
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   -   1   1   -   -   - \
     -   -   1 |.17
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   1   1   - \
     1   -   - |.18
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   1   -   1 \
     -   1   - |.19
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   1   1 \
     -   -   1 |.20
↦    1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   -   - \
     1   1   1 |.21
↦       === === === === === === === === === === === === === === === === === === === =
↦         2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2 \
     2   2   2
def H=grlifting3(R,S); grview(H);
↦             0     1
↦  ------------------
↦      1:    10     2
↦  ------------------
↦  total:    10     2
↦
↦             0     1     2     3
↦  ------------------------------
↦      0:     1     -     -     -
↦      1:    20    20    10     2
↦ ------------------------------
↦ total:    21    20    10     2
↦
↦ t:  2
↦ Graded homomorphism: r(-2)^20 <- r(-2)^2, given by a matrix, with degrees\
  :
↦      ..1 ..2 ....
↦      --- --- +...
↦   2 :  0   0 |..1
↦   2 :  0   0 |..2
↦   2 :  0   0 |..3
↦   2 :  0   0 |..4
↦   2 :  0   0 |..5
↦   2 :  0   0 |..6
↦   2 :  0   0 |..7
↦   2 :  0   0 |..8
↦   2 :  0   0 |..9
↦   2 :  0   0 |.10
↦   2 :  0   0 |.11
↦   2 :  0   0 |.12
↦   2 :  0   0 |.13
↦   2 :  0   0 |.14
↦   2 :  0   0 |.15
↦   2 :  0   0 |.16
↦   2 :  0   0 |.17
↦   2 :  0   0 |.18
↦   2 :  0   0 |.19
↦   2 :  0   0 |.20
↦      === ===
↦        2   2
↦ k:  1
↦ Graded homomorphism: r + r(-1)^20 <- r(-1)^10, given by a matrix, with de\
  grees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   0 :  -   -   -   -   -   -   -   -   -   - |..1
↦   1 :  0   0   0   -   -   0   0   0   -   - |..2
↦   1 :  0   0   -   0   -   0   0   -   0   - |..3
↦   1 :  0   0   -   -   0   0   0   -   -   0 |..4
↦   1 :  0   -   0   0   -   0   -   0   0   - |..5
↦   1 :  0   -   0   -   0   0   -   0   -   0 |..6
↦   1 :  0   -   -   0   0   0   -   -   0   0 |..7
↦   1 :  -   0   0   0   -   -   0   0   0   - |..8
↦   1 :  -   0   0   -   0   -   0   0   -   0 |..9
↦   1 :  -   0   -   0   0   -   0   -   0   0 |.10
↦   1 :  -   -   0   0   0   -   -   0   0   0 |.11
↦   1 :  0   0   0   -   -   0   0   0   -   - |.12
↦   1 :  0   0   -   0   -   0   0   -   0   - |.13
↦   1 :  0   0   -   -   0   0   0   -   -   0 |.14
↦   1 :  0   -   0   0   -   0   -   0   0   - |.15
↦   1 :  0   -   0   -   0   0   -   0   -   0 |.16
↦   1 :  0   -   -   0   0   0   -   -   0   0 |.17
↦   1 :  -   0   0   0   -   -   0   0   0   - |.18
↦   1 :   -   0   0   -   0   -   0   0   -   0 |.19
↦   1 :   -   0   -   0   0   -   0   -   0   0 |.20
↦   1 :   -   -   0   0   0   -   -   0   0   0 |.21
↦       === === === === === === === === === ===
↦         1   1   1   1   1   1   1   1   1   1
↦ Graded-object collection, given by the following maps (named here as o_[1\
   .. 2]):
↦ o_1 :
↦ Graded homomorphism: r + r(-1)^20 <- r(-1)^10, given by a matrix, with de\
  grees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦       --- --- --- --- --- --- --- --- --- --- +...
↦   0 :   -   -   -   -   -   -   -   -   -   - |..1
↦   1 :   0   0   0   -   -   0   0   0   -   - |..2
↦   1 :   0   0   -   0   -   0   0   -   0   - |..3
↦   1 :   0   0   -   -   0   0   0   -   -   0 |..4
↦   1 :   0   -   0   0   -   0   -   0   0   - |..5
↦   1 :   0   -   0   -   0   0   -   0   -   0 |..6
↦   1 :   0   -   -   0   0   0   -   -   0   0 |..7
↦   1 :   -   0   0   0   -   -   0   0   0   - |..8
↦   1 :   -   0   0   -   0   -   0   0   -   0 |..9
↦   1 :   -   0   -   0   0   -   0   -   0   0 |.10
↦   1 :   -   -   0   0   0   -   -   0   0   0 |.11
↦   1 :   0   0   0   -   -   0   0   0   -   - |.12
↦   1 :   0   0   -   0   -   0   0   -   0   - |.13
↦   1 :   0   0   -   -   0   0   0   -   -   0 |.14
↦   1 :   0   -   0   0   -   0   -   0   0   - |.15
↦   1 :   0   -   0   -   0   0   -   0   -   0 |.16
↦   1 :   0   -   -   0   0   0   -   -   0   0 |.17
↦   1 :   -   0   0   0   -   -   0   0   0   - |.18
↦   1 :   -   0   0   -   0   -   0   0   -   0 |.19
↦   1 :   -   0   -   0   0   -   0   -   0   0 |.20
↦   1 :   -   -   0   0   0   -   -   0   0   0 |.21
↦       === === === === === === === === === ===
↦         1   1   1   1   1   1   1   1   1   1
↦ o_2 :
↦ Graded homomorphism: r(-2)^20 <- r(-2)^2, given by a matrix, with degrees\
  :
↦       ..1 ..2 ....
↦       --- --- +...
↦   2 :   0   0 |..1
↦   2 :   0   0 |..2
↦   2 :   0   0 |..3
↦   2 :   0   0 |..4
↦   2 :   0   0 |..5
↦   2 :   0   0 |..6
↦   2 :   0   0 |..7
↦   2 :   0   0 |..8
↦   2 :   0   0 |..9
↦   2 :   0   0 |.10
↦   2 :   0   0 |.11
↦   2 :   0   0 |.12
↦   2 :   0   0 |.13
↦   2 :  0   0 |.14
↦   2 :  0   0 |.15
↦   2 :  0   0 |.16
↦   2 :  0   0 |.17
↦   2 :  0   0 |.18
↦   2 :  0   0 |.19
↦   2 :  0   0 |.20
↦      === ===
↦        2   2
// BUG in the proc
def G=mappingcone3(R,S);
↦ // ** redefining G (def G=mappingcone3(R,S);) ./examples/mappingcone3.sin\
   g:24
↦            0     1
↦ ------------------
↦     1:    10     2
↦ ------------------
↦ total:    10     2
↦ 
↦            0     1     2     3
↦ ------------------------------
↦     0:     1     -     -     -
↦     1:    20    20    10     2
↦ ------------------------------
↦ total:    21    20    10     2
↦ 
↦ t:  2
↦ Graded homomorphism: r(-2)^20 <- r(-2)^2, given by a matrix, with degrees\
   :
↦      ..1 ..2 ....
↦      --- --- +...
↦   2 :  0   0 |..1
↦   2 :  0   0 |..2
↦   2 :  0   0 |..3
↦   2 :  0   0 |..4
↦   2 :  0   0 |..5
↦   2 :  0   0 |..6
↦   2 :  0   0 |..7
↦   2 :  0   0 |..8
↦   2 :  0   0 |..9
↦   2 :  0   0 |.10
↦   2 :  0   0 |.11
↦   2 :  0   0 |.12
↦   2 :  0   0 |.13
↦   2 :  0   0 |.14
↦   2 :  0   0 |.15
↦   2 :  0   0 |.16
↦   2 :  0   0 |.17
↦   2 :  0   0 |.18
↦   2 :  0   0 |.19
↦   2 :  0   0 |.20
↦      === ===
↦        2   2
↦ k:  1
↦ Graded homomorphism: r + r(-1)^20 <- r(-1)^10, given by a matrix, with de\
  grees:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦       --- --- --- --- --- --- --- --- --- --- +...
↦    0 :  -   -   -   -   -   -   -   -   -   - |..1
↦    1 :  0   0   0   -   -   0   0   0   -   - |..2
↦    1 :  0   0   -   0   -   0   0   -   0   - |..3
↦    1 :  0   0   -   -   0   0   0   -   -   0 |..4
↦    1 :  0   -   0   0   -   0   -   0   0   - |..5
↦    1 :  0   -   0   -   0   0   -   0   -   0 |..6
↦    1 :  0   -   -   0   0   0   -   -   0   0 |..7
↦    1 :  -   0   0   0   -   -   0   0   0   - |..8
↦    1 :  -   0   0   -   0   -   0   0   -   0 |..9
↦    1 :  -   0   -   0   0   -   0   -   0   0 |.10
↦    1 :  -   -   0   0   0   -   -   0   0   0 |.11
↦    1 :  0   0   0   -   -   0   0   0   -   - |.12
↦    1 :  0   0   -   0   -   0   0   -   0   - |.13
↦    1 :  0   0   -   -   0   0   0   -   -   0 |.14
↦    1 :  0   -   0   0   -   0   -   0   0   - |.15
↦    1 :  0   -   0   -   0   0   -   0   -   0 |.16
↦    1 :  0   -   -   0   0   0   -   -   0   0 |.17
↦    1 :  -   0   0   0   -   -   0   0   0   - |.18
↦    1 :  -   0   0   -   0   -   0   0   -   0 |.19
↦    1 :  -   0   -   0   0   -   0   -   0   0 |.20
↦    1 :  -   -   0   0   0   -   -   0   0   0 |.21
↦       === === === === === === === === === ===
↦         1   1   1   1   1   1   1   1   1   1
↦ // ** redefining A (    module A=grconcat(P[i],rN[i]);) gradedModules.lib\
  ::mappingcone3:2370
↦ // ** redefining B (    module B=grobj(zero,v,w);) gradedModules.lib::map\
  pingcone3:2371
def I=KeneshlouMatrixPresentation(intvec(2,3,0,6,2));
def J=KeneshlouMatrixPresentation(intvec(4,0,1,2,1));
// def N=grlifting3(I,J);
// 2nd module does not lie in the first:
// def NN=mappingcone3(I,J); // ????????
```

D.15.6.32 grrange

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grrange(M), graded object M

Return: intvec

Purpose: get weights of module units, thus describing the target of M

Example:

```
LIB "gradedModules.lib";
ring r=32003,(x,y,z),dp;
module Z = grobj(freemodule(0),intvec(0:0),intvec(0:0));
grrange(Z);
↦
grdeg(Z);
```

```
↦
grview(Z);
↦ Graded homomorphism: 0 <- 0, given by zero (0^2) matrix.
module P=grobj(module([xy,0,xz]),intvec(0,1,0));
grrange(P);
↦ 0,1,0
grdeg(P);
↦ 2
grview(P);
↦ Graded homomorphism: r + r(-1) + r <- r(-2), given by a matrix, with degr\
   ees:
↦       ..1 ....
↦       --- +...
↦    0 :  2 |..1
↦    1 :  - |..2
↦    0 :  2 |..3
↦       ===
↦         2
```

D.15.6.33 grneg

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: grneg(A), graded object A

Return: graded object

Purpose: graded map defined by -A

Example:

```
LIB "gradedModules.lib";
ring r=0,(x,y,z),dp;
def A=grobj([x2,yz,xyz],intvec(1,1,0));
grview(A);
↦ Graded homomorphism: r(-1)^2 + r <- r(-3), given by a matrix, with degree\
   s:
↦     .1 ...
↦     -- +..
↦   1 : 2 |.1
↦   1 : 2 |.2
↦   0 : 3 |.3
↦     ==
↦      3
def F=grneg(A);
grview(F);
↦ Graded homomorphism: r(-1)^2 + r <- r(-3), given by a matrix, with degree\
   s:
↦     .1 ...
↦     -- +..
↦   1 : 2 |.1
↦   1 : 2 |.2
↦   0 : 3 |.3
↦     ==
↦      3
```

D.15.6.34 matrixpres

Procedure from library `gradedModules.lib` (see Section D.15.6 [gradedModules_lib], page 2345).

Usage: matrixpres(a), intvec a

Return: graded object

Purpose: matrix presentation for direct sum of omega^a[i](i) in form of a graded object

Example:

```
LIB "gradedModules.lib";
ring r = 32003,(x(0..4)),dp;
def R=matrixpres(intvec(1,4,0,0,0));
↦  // ** redefining j (     int j=size(a)-i;) gradedModules.lib::matrixpres\
   :2480
grview(R);
↦  Graded homomorphism: r(-1)^21 <- r(-2)^4, given by a matrix, with degrees\
   :
↦       ..1 ..2 ..3 ..4 ....
↦       --- --- --- --- +...
↦    1 :  -   -   -   - |..1
↦    1 :  1   -   -   - |..2
↦    1 :  1   -   -   - |..3
↦    1 :  1   -   -   - |..4
↦    1 :  1   -   -   - |..5
↦    1 :  1   -   -   - |..6
↦    1 :  -   1   -   - |..7
↦    1 :  -   1   -   - |..8
↦    1 :  -   1   -   - |..9
↦    1 :  -   1   -   - |.10
↦    1 :  -   1   -   - |.11
↦    1 :  -   -   1   - |.12
↦    1 :  -   -   1   - |.13
↦    1 :  -   -   1   - |.14
↦    1 :  -   -   1   - |.15
↦    1 :  -   -   1   - |.16
↦    1 :  -   -   -   1 |.17
↦    1 :  -   -   -   1 |.18
↦    1 :  -   -   -   1 |.19
↦    1 :  -   -   -   1 |.20
↦    1 :  -   -   -   1 |.21
↦       === === === ===
↦         2   2   2   2
def S=matrixpres(intvec(0,0,3,0,0));
↦  // ** redefining j (     int j=size(a)-i;) gradedModules.lib::matrixpres\
   :2480
grview(S);
↦  Graded homomorphism: r(-1)^30 <- r(-2)^15, given by a matrix, with degree\
   s:
↦       ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 .11 .12 .13 .14 .15 ....
↦       --- --- --- --- --- --- --- --- --- --- --- --- --- --- --- +...
↦    1 :  1   1   -   -   -   -   -   -   -   -   -   -   -   -   - |..1
↦    1 :  1   -   1   -   -   -   -   -   -   -   -   -   -   -   - |..2
↦    1 :  1   -   -   1   -   -   -   -   -   -   -   -   -   -   - |..3
↦   1 :  1   -   -   -   1   -   -   -   -   -   -   -   -   -   - |..4
↦   1 :  -   1   1   -   -   -   -   -   -   -   -   -   -   -   - |..5
↦   1 :  -   1   -   1   -   -   -   -   -   -   -   -   -   -   - |..6
↦   1 :  -   1   -   -   1   -   -   -   -   -   -   -   -   -   - |..7
↦   1 :  -   -   1   1   -   -   -   -   -   -   -   -   -   -   - |..8
↦   1 :  -   -   1   -   1   -   -   -   -   -   -   -   -   -   - |..9
↦   1 :  -   -   -   1   1   -   -   -   -   -   -   -   -   -   - |.10
↦   1 :  -   -   -   -   -   1   1   -   -   -   -   -   -   -   - |.11
↦   1 :  -   -   -   -   -   1   -   1   -   -   -   -   -   -   - |.12
↦   1 :  -   -   -   -   -   1   -   -   1   -   -   -   -   -   - |.13
↦   1 :  -   -   -   -   -   1   -   -   -   1   -   -   -   -   - |.14
↦   1 :  -   -   -   -   -   -   1   1   -   -   -   -   -   -   - |.15
↦   1 :  -   -   -   -   -   -   1   -   1   -   -   -   -   -   - |.16
↦   1 :  -   -   -   -   -   -   1   -   -   1   -   -   -   -   - |.17
↦   1 :  -   -   -   -   -   -   -   1   1   -   -   -   -   -   - |.18
↦   1 :  -   -   -   -   -   -   -   1   -   1   -   -   -   -   - |.19
↦   1 :  -   -   -   -   -   -   -   -   1   1   -   -   -   -   - |.20
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   1   -   -   - |.21
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   1   -   - |.22
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   -   1   - |.23
↦   1 :  -   -   -   -   -   -   -   -   -   -   1   -   -   -   1 |.24
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   1   -   - |.25
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   -   1   - |.26
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   1   -   -   1 |.27
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   1   1   - |.28
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   1   -   1 |.29
↦   1 :  -   -   -   -   -   -   -   -   -   -   -   -   -   1   1 |.30
↦      === === === === === === === === === === === === === === ===
↦        2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
def N1 = matrixpres(intvec(2,0,0,0,0));
grview(N1);
↦ Graded homomorphism: r(-1)^2 <- 0, given by zero (2 x 0) matrix.
def N2 = matrixpres(intvec(0,0,0,0,3));
grview(N2);
↦ Graded homomorphism: r^3 <- 0, given by zero (3 x 0) matrix.
def N = matrixpres(intvec(2,0,0,0,3));
grview(N);
↦ Graded homomorphism: r(-1)^2 + r^3 <- 0, given by zero (5 x 0) matrix.
def M1 = matrixpres(intvec(0,1,0,0,0));
↦ // ** redefining j (      int j=size(a)-i;) gradedModules.lib::matrixpres\
   :2480
grview(M1);
↦ Graded homomorphism: r(-1)^5 <- r(-2), given by a matrix, with degrees:
↦     .1 ...
↦     -- +..
↦  1 : 1 |.1
↦  1 : 1 |.2
↦  1 : 1 |.3
↦  1 : 1 |.4
↦  1 : 1 |.5
↦     ==
↦      2
def M2 = matrixpres(intvec(0,1,1,0,0));
```

```
↦ // ** redefining j (      int j=size(a)-i;) gradedModules.lib::matrixpres\
  :2480
↦ // ** redefining j (          int j=size(a)-i;) gradedModules.lib::matrix\
  pres:2488
grview(M2);
↦ Graded homomorphism: r(-1)^15 <- r(-2)^6, given by a matrix, with degrees\
  :
↦      ..1 ..2 ..3 ..4 ..5 ..6 ....
↦      --- --- --- --- --- --- +...
↦   1 :  1   -   -   -   -   - |..1
↦   1 :  1   -   -   -   -   - |..2
↦   1 :  1   -   -   -   -   - |..3
↦   1 :  1   -   -   -   -   - |..4
↦   1 :  1   -   -   -   -   - |..5
↦   1 :  -   1   1   -   -   - |..6
↦   1 :  -   1   -   1   -   - |..7
↦   1 :  -   1   -   -   1   - |..8
↦   1 :  -   1   -   -   -   1 |..9
↦   1 :  -   -   1   1   -   - |.10
↦   1 :  -   -   1   -   1   - |.11
↦   1 :  -   -   1   -   -   1 |.12
↦   1 :  -   -   -   1   1   - |.13
↦   1 :  -   -   -   1   -   1 |.14
↦   1 :  -   -   -   -   1   1 |.15
↦      === === === === === ===
↦        2   2   2   2   2   2
def M3 = matrixpres(intvec(0,0,0,1,0));
↦ // ** redefining j (      int j=size(a)-i;) gradedModules.lib::matrixpres\
  :2480
grview(M3);
↦ Graded homomorphism: r(-1)^10 <- r(-2)^10, given by a square matrix, with\
   degrees:
↦      ..1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 .10 ....
↦      --- --- --- --- --- --- --- --- --- --- +...
↦   1 :  1   1   -   -   1   -   -   -   -   - |..1
↦   1 :  1   -   1   -   -   1   -   -   -   - |..2
↦   1 :  1   -   -   1   -   -   1   -   -   - |..3
↦   1 :  -   1   1   -   -   -   -   1   -   - |..4
↦   1 :  -   1   -   1   -   -   -   -   1   - |..5
↦   1 :  -   -   1   1   -   -   -   -   -   1 |..6
↦   1 :  -   -   -   -   1   1   -   1   -   - |..7
↦   1 :  -   -   -   -   1   -   1   -   1   - |..8
↦   1 :  -   -   -   -   -   1   1   -   -   1 |..9
↦   1 :  -   -   -   -   -   -   -   1   1   1 |.10
↦      === === === === === === === === === ===
↦        2   2   2   2   2   2   2   2   2   2
def M = matrixpres(intvec(1,1,1,0,0));
↦ // ** redefining j (      int j=size(a)-i;) gradedModules.lib::matrixpres\
  :2480
↦ // ** redefining j (          int j=size(a)-i;) gradedModules.lib::matrix\
  pres:2488
grview(M);
↦ Graded homomorphism: r(-1)^16 <- r(-2)^6, given by a matrix, with degrees\
   :
↦       ..1 ..2 ..3 ..4 ..5 ..6 ....
↦       --- --- --- --- --- --- +...
↦    1 :  -   -   -   -   -   - |..1
↦    1 :  1   -   -   -   -   - |..2
↦    1 :  1   -   -   -   -   - |..3
↦    1 :  1   -   -   -   -   - |..4
↦    1 :  1   -   -   -   -   - |..5
↦    1 :  1   -   -   -   -   - |..6
↦    1 :  -   1   1   -   -   - |..7
↦    1 :  -   1   -   1   -   - |..8
↦    1 :  -   1   -   -   1   - |..9
↦    1 :  -   1   -   -   -   1 |.10
↦    1 :  -   -   1   1   -   - |.11
↦    1 :  -   -   1   -   1   - |.12
↦    1 :  -   -   1   -   -   1 |.13
↦    1 :  -   -   -   1   1   - |.14
↦    1 :  -   -   -   1   -   1 |.15
↦    1 :  -   -   -   -   1   1 |.16
↦       === === === === === ===
↦         2   2   2   2   2   2
```

D.15.7 hodge_lib

Library: hodge.lib

Purpose: Algorithms for Hodge ideals

Authors: Guillem Blanco, email: guillem.blanco@kuleuven.be

Overview: A library for computing the Hodge ideals [MP19] of Q-divisors associated to any reduced hypersurface $finR$.
The implemented algorithm [Bla21] is based on the characterization of the Hodge ideals in terms of the V -filtration of Malgrange and Kashiwara on $R_f f^s$, see [MP20].
As a consequence, this library provides also an algorithm to compute the multiplier ideals and the jumping numbers of any hypersurface, see [BS05].

References:

[Bla21] G. Blanco, An algorithm for Hodge ideals, to appear.
[BS05] N. Budur, M. Saito, Multiplier ideals, V-filtration, and spectrum, J. Algebraic Geom. 14 (2005), no. 2, 269-282. 2, 4
[MP19] M. Mustata, M. Popa: Hodge ideals, Mem. Amer. Math. Soc. 262 (2019), no. 1268
[MP20] M. Mustata, M. Popa: Hodge ideals for Q-divisors, V-filtration, and minimal exponent, Forum Math. Sigma 8 (2020), no. e19, 41 pp.

Procedures: See also: Section 7.5.5 [dmodapp_lib], page 420.

D.15.7.1 Vfiltration

Procedure from library `hodge.lib` (see Section D.15.7 [hodge_lib], page 2421).

Usage: Vfiltration(f, p [, eng]); f a poly, p a non-negative integer, eng an optional integer.

Return: ring

Purpose: compute R -generators for the V -filtration on $R_f f^s$ truncated up to degree p in ∂_t .

Note: activate the output ring with the `setring` command.
In the output ring, the list `Vfilt` contains the V -filtration.
The value of `eng` controls the algorithm used for Groebner basis computations.
See the `engine` procedure from Section 7.5.5 [dmodapp_lib], page 420 for the available algorithms.

Display: If `printlevel`=1, progress debug messages will be printed.

Example:

```
LIB "hodge.lib";
ring R = 0,(x,y),dp;
poly f = y^2-x^3;
def D = Vfiltration(f, 1);
setring D; Vfilt;
↦ [1]:
↦    [1]:
↦       [1]:
↦          -1
↦       [2]:
↦          -Dt*y
↦       [3]:
↦          -Dt*x
↦    [2]:
↦       1/6
↦    [3]:
↦       1
↦ [2]:
↦    [1]:
↦       [1]:
↦          -Dt*y
↦       [2]:
↦          -1
↦       [3]:
↦          -Dt*x^2
↦    [2]:
↦       5/6
↦    [3]:
↦       1
↦ [3]:
↦    [1]:
↦       [1]:
↦          -2*Dt*y^2+1
↦       [2]:
↦          -Dt*x*y
↦       [3]:
↦          -y
↦       [4]:
↦          -x
↦       [5]:
↦          -3*Dt*x^3-1
↦    [2]:
↦       1
↦     [3]:
↦        1
```

D.15.7.2 hodgeIdeals

Procedure from library `hodge.lib` (see Section D.15.7 [hodge_lib], page 2421).

Usage: hodgeIdeals(f, p [, eng]); f a reduced poly, p a non-negative integer, eng an optional integer.

Return: ring

Purpose: compute the Hodge ideals of f^α up to level p , for a reduced hypersurface f .

Note: activate the output ring with the `setring` command.
In the output ring, the list of ideals `hodge` contains the Hodge ideals of f .
The value of `eng` controls the algorithm used for Groebner basis computations.
See the `engine` procedure from Section 7.5.5 [dmodapp_lib], page 420 for the available algorithms.

Display: If `printlevel`=1, progress debug messages will be printed.

Example:

```
LIB "hodge.lib";
ring R = 0,(x,y),dp;
poly f = y^2-x^3;
def Ra = hodgeIdeals(f, 2);
setring Ra; hodge;
↦ [1]:
↦    [1]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=y
↦          _[2]=x
↦       [3]:
↦          _[1]=y^2
↦          _[2]=x*y
↦          _[3]=x^3
↦    [2]:
↦       1/6
↦    [3]:
↦       1
↦ [2]:
↦    [1]:
↦       [1]:
↦          _[1]=1
↦       [2]:
↦          _[1]=y
↦          _[2]=x^2
↦       [3]:
↦          _[1]=y^3
↦          _[2]=x*y^2
↦          _[3]=x^2*y
↦          _[4]=x^3+(2*a+1)*y^2
↦    [2]:
```

```
↦       5/6
↦    [3]:
↦       1
↦ [3]:
↦    [1]:
↦       [1]:
↦          _[1]=y
↦          _[2]=x
↦       [2]:
↦          _[1]=y^2
↦          _[2]=x*y
↦          _[3]=x^3
↦       [3]:
↦          _[1]=y^3
↦          _[2]=x^2*y^2
↦          _[3]=x^3*y
↦          _[4]=x^4+(2*a+1)*x*y^2
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.15.7.3 multIdeals

Procedure from library `hodge.lib` (see Section D.15.7 [hodge_lib], page 2421).

Usage: multIdeals(f, [, eng]); f a reduced poly, eng an optional integer.

Return: list

Purpose: compute the multiplier ideals of a hypersurface $f \in R$.

Note: The value of `eng` controls the algorithm used for Groebner basis computations. See the `engine` procedure from Section 7.5.5 [dmodapp_lib], page 420 for the available algorithms.

Display: If `printlevel`=1, progress debug messages will be printed.

Example:

```
LIB "hodge.lib";
ring R = 0,(x,y),dp;
poly f = y^2-x^3;
multIdeals(f);
↦ [1]:
↦    [1]:
↦       _[1]=1
↦    [2]:
↦       5/6
↦    [3]:
↦       1
↦ [2]:
↦    [1]:
↦       _[1]=y
↦       _[2]=x
↦    [2]:
↦       1
↦    [3]:
↦       1
```

D.15.7.4 nextHodgeIdeal

Procedure from library `hodge.lib` (see Section D.15.7 [hodge_lib], page 2421).

Usage: nextHodgeIdeal(f, I, p); f a poly, I an ideal, p a non-negative integer

Return: ideal

Purpose: given the p -th Hodge ideal I of f^α compute the $p+1$ -th Hodge ideal assuming that the Hodge filtration of the underlying mixed Hodge module is generated at level less than or equal to p .

Example:

```
LIB "hodge.lib";
ring R = 0,(x,y),dp;
poly f = y^2-x^3;
def Ra = hodgeIdeals(f, 2);
setring(Ra);
int p = 1;
nextHodgeIdeal(y^2-x^3, hodge[3][1][p+1], p);
↦ _[1]=x^2*y^2
↦ _[2]=y^3
↦ _[3]=x^3*y
↦ _[4]=x^4+(2*a+1)*x*y^2
```

D.15.8 lrcalc_lib

Library: lrcalc.lib

Purpose: An interface to the Littlewood-Richardson Calculator by Anders Buch

Author: Oleksandr Iena, o.g.yena@gmail.com

Overview: An interface to the documented functions of the Littlewood-Richardson Calculator by Anders Buch is implemented.
The library requires the Littlewood-Richardson Calculator by Anders Buch, which is available at http://math.rutgers.edu/~asbuch/lrcalc/

References:
[1] http://math.rutgers.edu/~asbuch/lrcalc/
http://math.rutgers.edu/~asbuch/lrcalc/lrcalc-1.2/README

Procedures:

D.15.8.1 LRinstall

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRinstall();

Return: int (exit status of the shell)

Purpose: installs the Littlewood-Richardson Calculator

Note:

Example:

```
LIB "lrcalc.lib";
// In order to install the Littlewood-Richardson Calculator
// type "LRinstall();"
// This will execute the following commands:
// wget math.rutgers.edu/~asbuch/lrcalc/lrcalc-1.2.tar.gz
// tar zxvf lrcalc-1.2.tar.gz
// cd lrcalc-1.2
// ./configure
// make
// sudo make install
```

D.15.8.2 LRcoef

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRcoef(z, x, y); z, x, y lists of integers (partitions)

Return: bigint

Purpose: computes the Littlewood-Richardson coefficient c^z_{x, y}

Note:

Example:

```
LIB "lrcalc.lib";
// Compute the Littlewood-Richardson coefficient c^z_{x, y}
// for z= (3, 2, 1), x=(2, 1), y=(2, 1)
list z = 3, 2, 1;
list x = 2, 1;
list y = 2, 1;
LRcoef(z, x, y);
↦ 2
```

D.15.8.3 LRskew

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRskew(z, x [,s, r]); z, x lists of integers (partitions) s string equal to 'r', r non-negative integer

Return: list of lists

Purpose: computes the partitions y for which the Littlewood-Richardson coefficient c^z_{x,y} is non-zero together with that coefficient; only partitions up to length r are computed if the optional parameters age given

Note:

Example:

```
LIB "lrcalc.lib";
// Compute the partitions y for which the Littlewood-Richardson coefficient
// c^z_{x,y} is non-zero together with that coefficient
// for z= (3, 2, 1), x=(2, 1)
list z = 3, 2, 1;
list x = 2, 1;
LRskew(z, x);
```

```
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦ [2]:
↦    [1]:
↦       2
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦       [3]:
↦          1
// Now compute only the partitions with at most 2 entries
LRskew(z, x, "r", 2);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦ [2]:
↦    [1]:
↦       2
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
```

D.15.8.4 LRmult

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRmult(x, y); x, y lists of integers (partitions)
LRmult(x, y [, s, r]); x, y lists of integers (partitions), s string equal to 'r', r integer
LRmult(x, y [, s, m, k]); x, y lists of integers (partitions), s string equal to 'q' or 'f', m, k integers

Return: list of lists

Purpose: computes the partitions z for which the Littlewood-Richardson coefficient c^z_{x,y} is non-zero together with that coefficient; partitions up to length r

Note:

Example:

```
LIB "lrcalc.lib";
// Compute the partitions z for which the Littlewood-Richardson coefficient
// c^z_{x,y} is non-zero together with that coefficient
// for x= (2, 1), y=(2, 1)
list x = 2, 1;
list y = 2, 1;
LRmult(x, y);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          3
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          4
↦       [2]:
↦          2
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦       [3]:
↦          1
↦       [4]:
↦          1
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦       [3]:
↦          2
↦ [5]:
↦    [1]:
↦       2
↦    [2]:
↦       [1]:
```

```
↦       3
↦     [2]:
↦       2
↦     [3]:
↦       1
↦ [6]:
↦   [1]:
↦     1
↦   [2]:
↦     [1]:
↦       4
↦     [2]:
↦       1
↦     [3]:
↦       1
↦ [7]:
↦   [1]:
↦     1
↦   [2]:
↦     [1]:
↦       3
↦     [2]:
↦       1
↦     [3]:
↦       1
↦     [4]:
↦       1
// Now compute only the partitions with at most 2 entries
LRmult(x, y, "r", 2);
↦ [1]:
↦   [1]:
↦     1
↦   [2]:
↦     [1]:
↦       3
↦     [2]:
↦       3
↦ [2]:
↦   [1]:
↦     1
↦   [2]:
↦     [1]:
↦       4
↦     [2]:
↦       2
// Now compute the product in the quantum cohomology ring of the Grassmannian Gr(3,3
LRmult(x, y, "q", 3, 2);
↦ [1]:
↦   [1]:
↦     1
↦   [2]:
↦     [1]:
↦       2
↦     [2]:
↦        2
↦     [3]:
↦        2
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          1
// Compute the same product with the output given in fusion ring notation
LRmult(x, y, "f", 3, 2);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦       [3]:
↦          2
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          2
↦       [3]:
↦          1
```

D.15.8.5 LRcoprod

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRcoprod(z); z list of integers (partition)

Return: list of lists

Purpose: computes the pairs of partitions x and y for which
the Littlewood-Richardson coefficient c^z_{x,y} is non-zero together with that coefficient

Note:

Example:

```
LIB "lrcalc.lib";
// Compute the pairs of partitions x and y for which the Littlewood-Richardson
// coefficient c^z_{x,y} is non-zero together with that coefficient
// for z= (3, 2, 1)
list z = 3, 2, 1;
LRcoprod(z);
↦ [1]:
```

```
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          1
↦    [3]:
↦       [1]:
↦          2
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦    [3]:
↦       [1]:
↦          3
↦ [5]:
↦    [1]:
↦       2
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦    [3]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦ [6]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          2
↦       [3]:
↦          1
↦    [3]:
↦       [1]:
↦          1
↦ [7]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          2
↦       [3]:
↦          1
↦    [3]:
↦       [1]:
↦          0
↦ [8]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          2
↦    [3]:
↦       [1]:
↦          1
↦ [9]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦       [3]:
```

```
↦          1
↦    [3]:
↦       [1]:
↦          2
↦ [10]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          1
↦       [3]:
↦          1
↦    [3]:
↦       [1]:
↦          1
↦ [11]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦       [3]:
↦          1
↦    [3]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦ [12]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          1
↦       [3]:
↦          1
↦    [3]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦ [13]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦          [2]:
↦             2
↦       [3]:
↦          [1]:
↦             2
```

D.15.8.6 LRschubmult

Procedure from library `lrcalc.lib` (see Section D.15.8 [lrcalc_lib], page 2425).

Usage: LRschubmult(x, y); x, y lists of integers

Return: list of lists

Purpose: computes the expantion of a product
of two Schubert polynomials in the basis of Schubert polynomials

Note:

Example:

```
LIB "lrcalc.lib";
// Compute the expantion of a square of the Schubert polynomial
// corresponding to  (1 3 2) in the basis of Schubert polynomials
list x = 1, 3, 2;
LRschubmult(x, x);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          1
↦       [2]:
↦          4
↦       [3]:
↦          2
↦       [4]:
↦          3
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          3
↦       [3]:
↦          1
```

D.15.9 maxlike_lib

Library: maxlike.lib

Purpose: Procedures to compute maximum likelihood estimates

Author: Adrian Koch (kocha at rhrk.uni-kl.de)

References:
Lior Pachter, Bernd Sturmfels; Algebraic Statistics for Computational Biology; published by Cambridge University Press

Procedures:

D.15.9.1 likeIdeal

Procedure from library `maxlike.lib` (see Section D.15.9 [maxlike_lib], page 2434).

Usage: likeIdeal(I,u); ideal I, intvec u
I represents the algebraic statistical model and u is the data vector under considerarion.

Return: ideal: the likelihood ideal with respect to I and u

Example:

```
LIB "maxlike.lib";
ring r = 0,(x,y),dp;
poly pA = -10x+2y+25;
poly pC = 8x-y+25;
poly pG = 11x-2y+25;
poly pT = -9x+y+25;
intvec u = 10,14,15,10;
ideal I = pA,pC,pG,pT;
ideal L = likeIdeal(I,u); L;
↦ L[1]=2744y2+13003050x-2116125y-6290625
↦ L[2]=9114xy-98y2+5828550x-891375y-2833125
↦ L[3]=847602x2-98y2+89716875x-12967425y-43985625
```

D.15.9.2 logHessian

Procedure from library `maxlike.lib` (see Section D.15.9 [maxlike_lib], page 2434).

Usage: logHessian(I,u); ideal I, intvec u
I represents the algebraic statistical model and u is the data vector under considerarion.

Return: matrix: a modified version of the Hessian matrix of the loglikelihood function defined by u and (the given generators of) I.

Note: This matrix has the following property: if it is negative definite at a point, then the actual Hessian is also negative definite at that point. The same holds for positive definiteness.

Example:

```
LIB "maxlike.lib";
ring r = 0,(x,y),dp;
poly pA = -10x+2y+25;
poly pC = 8x-y+25;
poly pG = 11x-2y+25;
poly pT = -9x+y+25;
intvec u = 10,14,15,10;
ideal I = pA,pC,pG,pT;
matrix H = logHessian(I,u); H;
↦ H[1,1]=-689040x3+314898x2y-44808xy2+1974y3+9619350x2-2949075xy+240975y2+6\
   151875x-1072500y-70640625
↦ H[1,2]=110880x3-51936x2y+7596xy2-348y3-1489200x2+431400xy-33450y2-1072500\
     x+176250y+11437500
↦ H[2,1]=110880x3-51936x2y+7596xy2-348y3-1489200x2+431400xy-33450y2-1072500\
     x+176250y+11437500
↦ H[2,2]=-16580x3+7972x2y-1192xy2+56y3+243150x2-66800xy+4900y2+176250x-2750\
     0y-1937500
```

D.15.9.3 getMaxPoints

Procedure from library `maxlike.lib` (see Section D.15.9 [maxlike_lib], page 2434).

Usage: getMaxPoints(Iu, H, prec [, "nodisplay"]); ideal Iu, matrix H, int prec, int k Iu the likelihood ideal, H the (modified) Hessian of the considered algebraic statistical model, prec the precision with which to compute the maximum likelihood estimates

Return: ring: a complex ring R in which you can find the following two lists: - MPOINTS, points in which the loglikelihood function has a local maximum, and - LHESSIANS, the (modified) Hessians at those points
also prints out the points in MPOINTS, unless a fourth argument is given

Note: it is assumed that the likelihood ideal is 0-dimensional

Example:

```
LIB "maxlike.lib";
ring r = 0,(x,y),dp;
poly pA = -10x+2y+25;
poly pC = 8x-y+25;
poly pG = 11x-2y+25;
poly pT = -9x+y+25;
intvec u = 10,14,15,10;
ideal I = pA,pC,pG,pT;
ideal L = likeIdeal(I,u);
matrix H = logHessian(I,u);
def R = getMaxPoints(L, H, 50);
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
↦ 
↦ // In the ring created by getmaxpoints you can find the lists
↦ //   MPOINTS, containing points in which the loglikelihood function has a\
   local maximum, and
↦ //   LHESSIANS, containing the (modified) Hessians at those points.
↦ 
setring R;
MPOINTS;
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
LHESSIANS;
↦ [1]:
↦    _[1,1]=-65487950.391931360088969690060635799847590217779318
↦    _[1,2]=10577428.579689959415257134363650588464921754723022
↦     _[2,1]=10577428.579689959415257134363650588464921754723022
↦     _[2,2]=-1795635.2877514452321365400508526832132830781483598
```

D.15.9.4 maxPoints

Procedure from library `maxlike.lib` (see Section D.15.9 [maxlike_lib], page 2434).

Usage: maxPoints(I,u,prec [, "nodisplay"]); ideal I, intvec u, int prec I represents the algebraic statistical model, u is the data vector under considerarion, and prec is the precision to be used in the computations

Return: ring: a complex ring R in which you can find the following two lists: - MPOINTS, points in which the loglikelihood function has a local maximum, and - LHESSIANS, the (modified) Hessians at those points
also prints out the points in MPOINTS, unless a fourth argument is given

Note: Just uses likeideal, loghessian and getmaxpoints.

Example:

```
LIB "maxlike.lib";
ring r = 0,(x,y),dp;
poly pA = -10x+2y+25;
poly pC = 8x-y+25;
poly pG = 11x-2y+25;
poly pT = -9x+y+25;
intvec u = 10,14,15,10;
ideal I = pA,pC,pG,pT;
def R = maxPoints(I, u, 50);
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
↦ 
↦ // In the ring created by getmaxpoints you can find the lists
↦ //   MPOINTS, containing points in which the loglikelihood function has a\
   local maximum, and
↦ //   LHESSIANS, containing the (modified) Hessians at those points.
↦ 
setring R;
MPOINTS;
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
LHESSIANS;
↦ [1]:
↦     _[1,1]=-65487950.391931360088969690060635799847590217779318
↦     _[1,2]=10577428.579689959415257134363650588464921754723022
↦     _[2,1]=10577428.579689959415257134363650588464921754723022
↦     _[2,2]=-1795635.2877514452321365400508526832132830781483598
```

D.15.9.5 maxPointsProb

Procedure from library `maxlike.lib` (see Section D.15.9 [maxlike_lib], page 2434).

Usage: maxPointsProb(I,u,prec [, "nodisplay"]); ideal I, intvec u, int prec I represents the algebraic statistical model, u is the data vector under considerarion, and prec is the precision to be used in the computations

Return: ring: a complex ring R in which you can find the following two lists: - MPOINTS, points in which the loglikelihood function has a local maximum, - LHESSIANS, the (modified) Hessians at those points, and - VALS, the resulting probability distributions (that is, the values of the polynomials given by I at the points in MPOINTS).
Also prints out the points in MPOINTS, unless a fourth argument is given.

Note: Does not compute the likelihood ideal via elimination, but rather computes the critical points by projection.

Example:

```
LIB "maxlike.lib";
ring r = 0,(x,y),dp;
poly pA = -10x+2y+25;
poly pC = 8x-y+25;
poly pG = 11x-2y+25;
poly pT = -9x+y+25;
intvec u = 10,14,15,10;
ideal I = pA,pC,pG,pT;
def R = maxPointsProb(I, u, 50);
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
↦ 
↦ // In the ring created by getmaxpoints you can find the lists
↦ //   MPOINTS, containing points in which the loglikelihood function has a\
   local maximum,
↦ //   LHESSIANS, containing the (modified) Hessians at those points, and
↦ //   VALS, containing the probability distributions at those points.
↦ 
setring R;
MPOINTS;
↦ [1]:
↦    [1]:
↦ 0.51912639453217837465463128685404418932771758896637
↦    [2]:
↦ 0.21725133256396491722887792998009835426225610459149
LHESSIANS;
↦ [1]:
↦    _[1,1]=-65487950.391931360088969690060635799847590217779318
↦    _[1,2]=10577428.579689959415257134363650588464921754723022
↦    _[2,1]=10577428.579689959415257134363650588464921754723022
↦    _[2,2]=-1795635.2877514452321365400508526832132830781483598
VALS;
↦ [1]:
↦    [1]:
↦       20.243238719806146087911442991419754815247336319519
↦    [2]:
↦       28.935759823693462080008172364852255160359484607139
↦     [3]:
↦        30.275887674726032286743188295434289374080381269447
↦     [4]:
↦        20.545113781774359545337196348293700650312797803894
```

D.15.10 modwalk_lib

Library: modwalk.lib

Purpose: Groebner basis conversion

Authors: S. Oberfranz oberfran@mathematik.uni-kl.de

Overview: A library for converting Groebner bases of an ideal in the polynomial ring over the rational numbers using modular methods. The procedures are inspired by the following paper:
Elizabeth A. Arnold: Modular algorithms for computing Groebner bases. Journal of Symbolic Computation 35, 403-419 (2003).

Procedures: See also: Section D.4.10 [grwalk_lib], page 1087; Section D.15.16 [rwalk_lib], page 2536; Section D.15.20 [swalk_lib], page 2550.

D.15.10.1 modWalk

Procedure from library `modwalk.lib` (see Section D.15.10 [modwalk_lib], page 2439).

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.

Example:

```
LIB "modwalk.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
I = std(I);
ring R2 = 0, (x,y,z,t), lp;
ideal I = fetch(R1, I);
ideal J = modWalk(I);
J;
↦ J[1]=x3+1/3x2+1/3
↦ J[2]=z4+1/5z2+4/5
↦ J[3]=y5+1/11y3+2/11
ring S1 = 0, (a,b,c,d), Dp;
ideal I = 5b2, ac2+9d3+3a2+5b, 2a2c+7abd+bcd+4a2, 2ad2+6b2d+7c3+8ad+4c;
I = std(I);
ring S2 = 0, (c,d,b,a), lp;
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a lp-Groebner basis.
ideal J = modWalk(I,"Dp");
J;
↦ J[1]=a25+16a24+96a23+256a22+256a21+256/9a20+1024/3a19+2048a18+65536/9a17+\
   32768/3a16+16384/81a15+131072/81a14+1048576/81a13+1048576/27a12+1048576/9\
   a11
↦ J[2]=ba11+1522867351997104938459/91668001658017308797687087104a24+4293036\
   9782248629690765/91668001658017308797687087104a23+80925218629630777478637\
   /22917000414504327199421771776a22+7108535670237178684767/2864625051813040\
   899927721472a21-3255817194541612658349/89519532869157528122741296a20+5380\
   8965391546362724459/358078131476630112490965184a19+1534729815590907963215\
   01/358078131476630112490965184a18-260815719913165309506063/44759766434578\
   764061370648a17-1485276141860757031491027/89519532869157528122741296a16-4\
   92332725360316960775/22379883217289382030685324a15+7423992361030571232440\
   /16784912412967036523013993a14-17640364913371983121693/167849124129670365\
   23013993a13-37723213977586186442564/5594970804322345507671331a12+92047580\
   41857159721472414/5594970804322345507671331a11
↦ J[3]=b2a6-63/2ba10+41087306587333357057895823883/924013456712814472680685\
   83800832a24+93915562116924232413944264677/1320019223875449246686694054297\
   6a23+1314662746341964624103002499857/30800448557093815756022861266944a22+\
   2628268795042931967685617407557/23100336417820361817017145950208a21+32861\
   7969148352577032114618159/2887542052227545227127143243776a20+175631724829\
   284757906915538269/12993939235023953522072144596992a19+303812212296039043\
   29090032467/206253003730538944794795945984a18+581332950565269518541458030\
   9/6445406366579342024837373312a17+1392577308804410648719627124495/4060606\
   01094498547564754518656a16+728489619081836101063651608791/135353533698166\
   182521584839552a15+80182830319998353431517995037/913636352462621732020697\
   666976a14+22597043807001043905240513127/32629869730807919000739202392a13+\
   2699689128255025271541196980091/456818176231310866010348833488a12+6291773\
   289016174025274735397/352483160672307766983293853a11+4/7a6
↦ J[4]=b3a5+4/7ba5-5398327059462849163101023479/739210765370251578144548670\
   406656a24+3500067651845908053488406611/92401345671281447268068583800832a2\
   3+81936880336538263803253409291/46200672835640723634034291900416a22+85587\
   506267677081700930548967/6600096119377246233433470271488a21+4349147060430\
   84846670081811347/11550168208910180908508572975104a20+1021713731964491721\
   016405426709/25987878470047907044144289193984a19+189087309367688338214840\
   3875/812121202188997095129509037312a18+3873208423822126582196454031/10312\
   6501865269472397397972992a17+104243440049859097996976327663/4060606010944\
   98547564754518656a16+52530075262606982469983545909/5075757513681231844559\
   4314832a15+6154556265260978917193662164647/365454540985048692808279066790\
   4a14+138400848395446486358821423/5639730570756924271732701648a13+68577767\
   53456717192310999393/38068181352609238834195736124a12+3439955547980759942\
   57860510037/228409088115655433005174416744a11+9/2a10+18a9
↦ J[5]=b4-63/2b3a4+4/7b2+3859043113737/128ba10-61261515/8ba9-525086793/32ba\
   8+3969/16ba7-19845/4ba6+9/2ba5-317530772199391516703685862633925319890715\
   5/9853890705461273608546303647569412096a24-889000025950798002413919214249\
   17407047623059/17244308734557228814956031383246471168a23-4443358152931308\
   1841158459544460024461570519/1437025727879769067913002615270539264a22-443\
   94201884268758058296712245149507940575423/5388846479549134004673759807264\
   52224a21-88794706503699382133468723144783169844707611/1077769295909826800\
   934751961452904448a20-1763218978464215084468461907410740543335425/1732129\
   22556936450150227993804931072a19-4436403904696107325808495191699219229446\
   85/4210036312147760941151374849425408a18-21791665871565292166347154127869\
   734967876143/33680290497182087529210998795403264a17-382244791239220889094\
   021792022832016196677083/151561307237319393881449494579314688a16-25381669\
   887869257097552146559952295821945667/6315054468221641411727062274138112a1\
   5-3081988985408645474889243822837753592680251/487161344691383766047516232\
   57636864a14-42991086686519981772739427292780808208480959/8525323532099215\
   9058315340700864512a13-44791324614077928834488691100822888689474979/10656\
   654415124019882289417587608064a12-250969820105414481651193596059252574282\
   0133/1973454521319262941164706960668l6a11+771901337907/64a10-1701/4a9+437\
   53689/4a8-19845/8a7-19845/2a6
↦ J[6]=da-1323/800b3a4+63/20b2a4+2701701074331/1280ba10-61269453/320ba9+306\
   291699/160ba8-27783/16ba6-189/200ba4-685883099436497069901143509465258298\
   0621641/123173633818265920106828795594617651200a24-5486790468650205133145\
   8584531975294116209709/61586816909132960053414397797308825600a23-60954530\
   98260341081175657932145681344303369/1140496609428388149137303662913126400\
   a22-54839253809480536502430419967744127237862429/384917605682081000333839\
   9862331801600a21-6855741428155530311932580353074238311128479/481147007102\
   601250417299982791475200a20-2878029541644786739384025698911163129301521/1\
   732129225569364501502279938049310720a19-539637460347386518746735121876349\
   5466944867/288688204261560750250379989674885120a18-1360851075145362976080\
   2370233473935440353147/120286751775650312604324995697868800a17-2261317453\
   3404843047583509526614944559001629/54129038299042640671946248064040960a16\
   -19899168362053216404911620136404443756991/309379505595808417192193918976\
   00a15-3443170203569948104100043669593479708510001/30447584043211485377969\
   7645360230400a14-27203224432640761938740031635681832222207391/30447584043\
   2114853779697645360230400a13-11007706754707314979975768772869745826795967\
   3/152237920216057426889848822680115200a12-1380727908059280736173048219943\
   2331439121223/6343246675669059453743700945004800a11-1929592445193/640a10+\
   437570343/1600a9+164090367/100a8+3969/8a6+9/5a4
↦ J[7]=db2+4/7d+81/160b3a4-189/200b3a3+9/5b2a3+250047/32ba10+4750893/320ba9\
   -567/40ba8-567/10ba7+81/280ba4-27/50ba3-627430952592078879073202920720376\
   153131/5173292620367168644486809414973941350400a24-2906859466655459274817\
   4299538370243493/18476045072739888016024319339192647680a23-69048240138740\
   571289193103124930690109/11975214398998075565941688460587827200a22+488085\
   08639174871490992992380443024047/11547528170462430010015199586995404800a2\
   1+5093069077149293873574182379116672811143/808326971932370100701063971089\
   67833600a20+3286117544936595300844164893345508176887/36374713736956654531\
   547878699035525120a19-57124126734867585149642894515011901057/189451634046\
   6492423518118682241433600a18-296407013643924816106633846582502767991/2526\
   021787288656564690824909655244800a17-193831832299525900077669479610197531\
   4379/11367098042798954541108712093448601600a16+49045157132492520042165362\
   0398209599/433131307834131784069071486566400a15+9565278519988924726047202\
   0815175898220317/25575970596297647717494602210259353600a14-51800242230268\
   556716933919846651112061/456713760648172280669546468040345600a13-15498981\
   08498974500694541229046869490537/1598498162268602982343412638141209600a12\
   -700479417253540849801700529315888259/26641636037810049705723543969020160\
   a11+20611017/1600a10+21504771/320a9+243/50a8-1701/100a7+36/35a3
↦ J[8]=d2
↦ J[9]=ca-1630191535/8275968d2ba2-36060437345/409660416d2ba+38107475/568972\
   8d2a2+4391588006225/10115072d2a+92575/2844864d2+91285075/182071296db4a+35\
   /9216db4-1088785/5689728db3a2+91259683/91035648db3a+24027/2528768db3-8162\
   999/79656192db2a3-6539063/79656192db2a2+25495277/45517824db2a+5095/126438\
   4db2-5243/158048dba4-109809/632192dba3-173005/632192dba2+265684003/159312\
   384dba+24027/4425344db-7/256da5+5/96da3+5/24da2+35/32da+2645/2489256d+861\
   7/1896576b5a-25921/158048b5+7/256b4a2-2163/79024b4a+2163/19756b4+529/3160\
   96b3a3+7/128b3a2-12811/59268b3a+15435/19756b3+311787/632192b2a4+529/15804\
   8b2a3+1/64b2a2-309/19756b2a+309/4939b2+19467/9878ba4+529/553168ba3+1/32ba\
   2-1/8ba+1/2b+3/32a6+3/8a5+63/32a4+529/276584a3
↦ J[10]=cb-452831001/4597760d2ba2-3340189909/75863040d2ba-3176523/158048d2a\
   2+2194845076291/10115072d2a+5145/316096d2+25404981/101150720db4a-7203/102\
   4db4+453789/790240db3a2+32184817/10115072db3a-25399101/12643840db3-324051\
   /6321920db2a3+194523/790240db2a2+11921259/25287680db2a-9921275/1264384db2\
   -27783/632192dba4-966217/3160960dba3-43211/395120dba2+1278655/2528768dba-\
   31286843/3160960db-5/48da2-5/12da-172823/79024d+64827/3160960b5a-1351/148\
   170b5+64827/1580480b4a-64827/395120b4+1323/1580480b3a3+9261/790240b3a-386\
   /74085b3-583443/395120b2a4-10887849/1580480b2a3+9261/395120b2a-9261/98780\
   b2-583443/197560ba4-6222951/395120ba3-3/16a5-3/4a4-1555407/395120a3
↦ J[11]=cd+66199/8d2ba2+6818809/2304d2ba+35/64d2b+47/96d2a2-7004232463/384d\
   2a-7/192db5+7/256db4a+7/320db4-1/80db3a2+3/640db3a-23/96db3+441/128db2a3+\
   1/64db2a+1/80db2+63/8dba3+1/32dba-1/8db+141/160da5+141/40da4+63/32da3-63/\
   80b4a2-63/40b3a2-9/20b2a2-9/10ba2
↦ J[12]=c2+1/3cb2+5/3d+3a3
intvec w = 3,2,1,2;
ring S3 = 0, (c,d,b,a), (a(w),lp);
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a (a(w),lp)-Groebner basis.
ideal J = modWalk(I,"Dp",w);
J;
↦ J[1]=d2
↦ J[2]=c2+3a3+1/3cb2+5/3d
↦ J[3]=ca2+4ca+7/2b3+2b
↦ J[4]=cda-6/7ba3+4/7c2-2/21cb3+1/7dba-10/21db
↦ J[5]=db4a-192/49b2a3+128/49c2b-64/7cda-64/147cb4+4/63d2ba+2db3a-4db4+60/4\
   9db2a-8db3+8/7dba-656/147db2-32/7db
↦ J[6]=db6+333576da3-189/4b8+15876b4a2-5186640/2401b2a3-2860453892752/17294\
   403c2b+119097010895/311299254cd2+1815960/49cdb2-1455130425704/2470629cda+\
   8395442084/7203cb4-2304/49cb2a-826877376/2401ca2+6069139793/155649627d2ba\
   -43215/7db5+199982018/49db3a-4664160/7dba2-126b7+72/7b5a+31752b3a2+1728/2\
   401ba3-5770728573920/17294403c2-1012272/49cdb+5598114168/2401cb3-4608/49c\
   ba+604630/21609d2b2-129552/49db4+864216/2401db2a-117b6+144/7b4a+9072b2a2-\
   20160cd+4663152/7cb2-3307509504/2401ca-7757580/16807d2b-172796/49db3+3888\
   0529598704/17294403dba-1296/7b5+288/49b3a+18144ba2+9326304/7cb+185320d2-1\
   9529120/7203db2-1188/7b4+576/49b2a+2752/2401db-413460864/343b3-3312/49b2-\
   1653754752/2401b
↦ J[7]=cb3a-5/8da3-245/1024b8-7/32b6a+2/343b2a3-945270062561/1067311728c2b+\
   595485054475/307385777664cd2+5/112cdb2-945263061245/304946208cda-108013/4\
   9392cb4+25/7cb2a-966391/1372ca2+63120825485/7684644416d2ba-3/256db5-1543\
   1/2016db3a+5/4dba2-245/256b7-25/32b5a+5/1372ba3-945267210173/533655864c2+\
   5/84cdb-108035/24696cb3+16/7cba+1512605/3556224d2b2-47/2688db4+53/24696db\
   2a-91/256b6-5/8b4a-5/84cd-5/4cb2-966979/343ca-3232325/1382976d2b+1/192db3\
   -954609818105/2134623456dba+65/32b5-1/14b3a-5/2cb-25/72d2-667/98784db2+85\
   /64b4-2/7b2a+109/12348db-1932803/784b3+3/14ba+93/112b2-966979/686b
↦ J[8]=cb5-4/7db2a2+96/7b2a3-64/7c2b-4/441cdb2+32cda+116/21cb4+1152/7ca2-8/\
   7dba2-8/441cdb+32/7cb3+14db4-16/7db2a+16/7cb2+4608/7ca+28db3+16/7cb+328/2\
   1db2+16db+576b3+2304/7b
↦ J[9]=cdb3+18da3+2cdb2+63cb4+441/2db3a-36dba2+126cb3+36cb2+126dba+72cb+10d\
   2
↦ J[10]=ba4-2/3c2a+1/9cb3a+14/3cda-1/6dba2+49/12db3+5/9dba+7/3db
↦ J[11]=b3a3+2744/5c2d+382/15c2b2-1127/15cdb3+7357/45cdba+129659/45cb5+2352\
   /5cb3a+906269/90db4a-8232/5db2a2-1029/10b6a+236/5c2b+161/5cdb2+5474/45cda\
   +28812/5cb4+1680cb2a+55174/405d2ba-98/15db5-2549/90db3a-735/2b5a-152/15c2\
   -224/15cdb+8232/5cb3+5376/5cba+2156/45db4+259588/45db2a+2058/5b6-294b4a-2\
   8cd+16464/5cb2-4032/5ca+1069/9db3-898/45dba+1470b5-168/5b3a+1232/45db2+11\
   76b4-672/5b2a+3136/45db+672/5b3+504/5ba+2688/5b2-2016/5b
↦ J[12]=b5a2+182/3c2d+32/9c2b2-71/9cdb3+28204/1323cdba+931/2cb5+64cb3a-116/\
   7cba2+17/1134d2b4+34/200120949d2b2a-7/18db6+175759/108db4a-117310/441db2a\
   2-14b6a+4b4a2+16480960/3176523c2b-340/200120949cd2+194/63cdb2+4999816/453\
   789cda+1225cb4+1600/7cb2a-128/7ca2+8819321851/400241898d2ba-11/6db5+19380\
   5/189db3a-74096/441dba2-50b5a+32/7b3a2-12215296/3176523c2-124/21cdb+854cb\
   3+80cba+34/3969d2b2+1079/189db4+1232881/1323db2a+56b6-40b4a+16/7b2a2-400/\
   63cd+700cb2-1280/7ca+406/27db3+1857442660/3176523dba+200b5-32/7b3a+16/7ba\
   2+336cb+4484/1323db2+102b4-128/7b2a+12160/1323db-320/7b3+96/7ba+40b2-640/\
   7b
↦ J[13]=b7a-13606662/245c2d+20584/15435c2b2+23004293/2205cdb3+865156/5145cd\
   ba+758/5cb5+3904/245cb3a+12618cba2+45280/9261d2b2a+4/63db6+8178503/15435d\
   b4a-3032/35db2a2-52/35b6a-1008b4a2+8917192/108045c2b+20176/15435cdb2+3119\
   812/5145cda+20430577/35cb4+26496/343cb2a+9216ca2+147043/19845d2ba+863872/\
   2205db5+7003823691/3430db3a-16335524/49dba2-4b7-772/49b5a-2016b3a2+172580\
   32/108045c2+2892256/2205cdb+40842962/35cb3+12382792/245cba+53656/315db4+5\
   20568/1715db2a+208/35b6-736/49b4a-576b2a2+3946912/3087cd+81722308/245cb2+\
   63150592/1715ca+100/441d2b+709600/3087db3+42025695302/36015dba+3088/49b5-\
   432/1715b3a-1152ba2+32671944/49cb+1502048/15435db2+2166931/49b4-1984/245b\
   2a+7552/2205db+55320768/1715b3+8896/1715ba+6190756/245b2+31575296/1715b
↦ J[14]=b9-48392128/5145c2d+60497909109056/16343210835c2b2-238194021790/294\
   17779503cd2b+8065888/5145cdb3+30249615617056/2334744405cdba+384/35cb5+307\
   2/1715cb3a-11648320/16807cba2-232497742664/29417779503d2b2a-4/441db6+1380\
   928/36015db4a-1536/245db2a2+4b8-96/245b6a-256/16807b2a3+119801054187904/1\
   6343210835c2b+7936/36015cdb2-3065728/12005cda+74726439712/756315cb4+16384\
   /2401cb2a-2420168/1361367d2b3+87904/138915d2ba-664/15435db5+12451636912/3\
   6015db3a-19361152/343dba2+36/7b7-512/343b5a-110595584/756315c2+256/108045\
   cdb+48404416/245cb3-232478976/84035cba+10343440/1058841d2b2+304/1715db4+3\
   0607186827808/16343210835db2a+1504/245b6-512/343b4a-1024/21609cd+96827264\
   /1715cb2-16384/12005ca-640/3087d2b+1056/2401db3+49797330496/252105dba+372\
   8/343b5-6656/12005b3a+38722304/343cb+72448/756315db2-5806688/2401b4-8704/\
   12005b2a+4096/15435db+42304/12005b3+2048/12005ba-116239488/84035b2-8192/1\
   2005b
↦ J[15]=dba3-2/3c2d+1/9cdb3-1/6d2ba+5/9d2b
↦ J[16]=db3a2-147/2cdb3-10cdba-288cba2+63b4a2-32c2b-132cda-576ca2-49/2db5+4\
   db3a+126b3a2-64c2-82cdb-1152cba-21/2db4+36b2a2-80cd-2304ca-14db3-16dba+72\
   ba2-6db2-1008b4-2016b3-576b2-1152b
↦ J[17]=a5+1/9cb2a2-4/3c2a-7/6cb3+5/9da2-2/3cb
↦ J[18]=da4-4/21c3+11/63cdba+8/63c2b-1/63db3a+10/63cdb+32/7cba+5/9d2a+2/63d\
   b2a-b4a+4b4-4/7b2a+16/7b2
```

See also: Section D.2.6.1 [modular], page 883.

D.15.10.2 modrWalk

Procedure from library `modwalk.lib` (see Section D.15.10 [modwalk_lib], page 2439).

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.

Example:

```
LIB "modwalk.lib";
ring R1 = 0, (x,y,z,t), dp;
```

```
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
I = std(I);
ring R2 = 0, (x,y,z,t), lp;
ideal I = fetch(R1, I);
int radius = 2;
ideal J = modrWalk(I,radius);
J;
↦ J[1]=x3+1/3x2+1/3
↦ J[2]=z4+1/5z2+4/5
↦ J[3]=y5+1/11y3+2/11
ring S1 = 0, (a,b,c,d), Dp;
ideal I = 5b2, ac2+9d3+3a2+5b, 2a2c+7abd+bcd+4a2, 2ad2+6b2d+7c3+8ad+4c;
I = std(I);
ring S2 = 0, (c,d,b,a), lp;
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a lp-Groebner basis.
ideal J = modrWalk(I,radius,"Dp");
J;
↦ J[1]=a25+16a24+96a23+256a22+256a21+256/9a20+1024/3a19+2048a18+65536/9a17+\
   32768/3a16+16384/81a15+131072/81a14+1048576/81a13+1048576/27a12+1048576/9\
   a11
↦ J[2]=ba11+1522867351997104938459/91668001658017308797687087104a24+4293036\
   9782248629690765/91668001658017308797687087104a23+80925218629630777478637\
   /22917000414504327199421771776a22+7108535670237178684767/2864625051813040\
   899927721472a21-3255817194541612658349/89519532869157528122741296a20+5380\
   8965391546362724459/358078131476630112490965184a19+1534729815590907963215\
   01/358078131476630112490965184a18-260815719913165309506063/44759766434578\
   764061370648a17-1485276141860757031491027/89519532869157528122741296a16-4\
   92332725360316960775/22379883217289382030685324a15+7423992361030571232440\
   /16784912412967036523013993a14-17640364913371983121693/167849124129670365\
   23013993a13-37723213977586186442564/5594970804322345507671331a12+92047580\
   41857159721472414/5594970804322345507671331a11
↦ J[3]=b2a6-63/2ba10+41087306587333357057895823883/924013456712814472680685\
   83800832a24+93915562116924232413944264677/1320019223875449246686694054297\
   6a23+1314662746341964624103002499857/30800448557093815756022861266944a22+\
   2628268795042931967685617407557/23100336417820361817017145950208a21+32861\
   7969148352577032114618159/2887542052227545227127143243776a20+175631724829\
   284757906915538269/12993939235023953522072144596992a19+303812212296039043\
   29090032467/206253003730538944794795945984a18+581332950565269518541458030\
   9/6445406366579342024837373312a17+1392577308804410648719627124495/4060606\
   01094498547564754518656a16+728489619081836101063651608791/135353533698166\
   182521584839552a15+80182830319998353431517995037/913636352462621732020697\
   666976a14+22597043807001043905240513127/32629869730807919000739202392a13+\
   2699689128255025271541196980091/456818176231310866010348833488a12+6291773\
   289016174025274735397/352483160672307766983293853a11+4/7a6
↦ J[4]=b3a5+4/7ba5-5398327059462849163101023479/739210765370251578144548670\
   406656a24+3500067651845908053488406611/92401345671281447268068583800832a2\
   3+81936880336538263803253409291/46200672835640723634034291900416a22+85587\
   506267677081700930548967/6600096119377246233433470271488a21+4349147060430\
   84846670081811347/11550168208910180908508572975104a20+1021713731964491721\
   016405426709/25987878470047907044144289193984a19+189087309367688338214840\
   3875/812121202188997095129509037312a18+3873208423822126582196454031/10312\
   6501865269472397397972992a17+104243440049859097996976327663/4060606010944\
   98547564754518656a16+52530075262606982469983545909/5075757513681231844559\
   4314832a15+6154556265260978917193662164647/365454540985048692808279066790\
   4a14+138400848395446486358821423/5639730570756924271732701648a13+68577767\
   53456717192310999393/38068181352609238834195736124a12+3439955547980759942\
   57860510037/228409088115655433005174416744a11+9/2a10+18a9
 ↦ J[5]=b4-63/2b3a4+4/7b2+3859043113737/128ba10-61261515/8ba9-525086793/32ba\
   8+3969/16ba7-19845/4ba6+9/2ba5-317530772199391516703685862633925319890715\
   5/9853890705461273608546303647569412096a24-889000025950798002413919214249\
   17407047623059/17244308734557228814956031383246471168a23-4443358152931308\
   1841158459544460024461570519/1437025727879769067913002615270539264a22-443\
   94201884268758058296712245149507940575423/5388846479549134004673759807264\
   52224a21-88794706503699382133468723144783169844707611/1077769295909826800\
   934751961452904448a20-1763218978464215084468461907410740543335425/1732129\
   22556936450150227993804931072a19-4436403904696107325808495191699219229446\
   85/4210036312147760941151374849425408a18-21791665871565292166347154127869\
   734967876143/33680290497182087529210998795403264a17-382244791239220889094\
   021792022832016196677083/151561307237319393881449494579314688a16-25381669\
   887869257097552146559952295821945667/6315054468221641411727062274138112a1\
   5-3081988985408645474889243822837753592680251/487161344691383766047516232\
   57636864a14-42991086686519981772739427292780808208480959/8525323532099215\
   9058315340700864512a13-44791324614077928834488691100822888689474979/10656\
   654415124019882289417587608064a12-250969820105414481651193596059252574282\
   0133/197345452131926294116470696066816a11+771901337907/64a10-1701/4a9+437\
   53689/4a8-19845/8a7-19845/2a6
 ↦ J[6]=da-1323/800b3a4+63/20b2a4+2701701074331/1280ba10-61269453/320ba9+306\
   291699/160ba8-27783/16ba6-189/200ba4-685883099436497069901143509465258298\
   0621641/123173633818265920106828795594617651200a24-5486790468650205133145\
   8584531975294116209709/61586816909132960053414397797308825600a23-60954530\
   98260341081175657932145681344303369/1140496609428388149137303662913126400\
   a22-54839253809480536502430419967744127237862429/384917605682081000333839\
   9862331801600a21-6855741428155530311932580353074238311128479/481147007102\
   601250417299982791475200a20-2878029541644786739384025698911163129301521/1\
   732129225569364501502279938049310720a19-539637460347386518746735121876349\
   5466944867/288688204261560750250379989674885120a18-1360851075145362976080\
   2370233473935440353147/120286751775650312604324995697868800a17-2261317453\
   3404843047583509526614944559001629/54129038299042640671946248064040960a16\
   -19899168362053216404911620136404443756991/309379505595808417192193918976\
   00a15-3443170203569948104100043669593479708510001/30447584043211485377969\
   7645360230400a14-27203224432640761938740031635681832222207391/30447584043\
   2114853779697645360230400a13-11007706754707314979975768772869745826795967\
   3/152237920216057426889848822680115200a12-1380727908059280736173048219943\
   2331439121223/6343246675669059453743700945004800a11-1929592445193/640a10+\
   437570343/1600a9+164090367/100a8+3969/8a6+9/5a4
 ↦ J[7]=db2+4/7d+81/160b3a4-189/200b3a3+9/5b2a3+250047/32ba10+4750893/320ba9\
   -567/40ba8-567/10ba7+81/280ba4-27/50ba3-627430952592078879073202920720376\
   153131/5173292620367168644486809414973941350400a24-2906859466655459274817\
   4299538370243493/18476045072739888016024319339192647680a23-69048240138740\
   571289193103124930690109/11975214398998075565941688460587827200a22+488085\
   08639174871490992992380443024047/11547528170462430010015199586995404800a2\
   1+5093069077149293873574182379116672811143/808326971932370100701063971089\
   67833600a20+3286117544936595300844164893345508176887/36374713736956654531\
   547878699035525120a19-57124126734867585149642894515011901057/189451634046\
   6492423518118682241433600a18-29640701364392481610663384658250276799 1/2526\
   0217872886565646908249096552448 00a17-1938318322995259000776694796101975 31\
   4379/1136709804279895454110871209344860 1600a16+4904515713249252004216536 2\
   0398209599/4331313078341317840690714865664 00a15+9565278519988924726047202\
   0815175898220317/25575970596297647717494602210259353600a14-51800242230268\
   5567169339198466511120 61/45671376064817228066954646804034 5600a13-15498981\
   0849897450069454122904686949 0537/15984981622686029823434126381412096 00a12\
   -7004794172535408498017005293158882 59/266416360378100497057235439690201 60\
   a11+20611017/1600a10+21504771/320a9+243/50a8-1701/100a7+36/35a3
↦  J[8]=d2
↦  J[9]=ca-1630191535/8275968d2ba2-36060437345/409660416d2ba+38107475/568972\
   8d2a2-456425375/60690432d2a+92575/2844864d2+91285075/182071296db4a+35/921\
   6db4-1088785/5689728db3a2+2258752616171/182071296db3a+24027/2528768db3-81\
   62999/79656192db2a3-6539063/79656192db2a2+25495277/45517824db2a+5095/1264\
   384db2-5243/158048dba4-753114969703/1486915584dba3-753467060575/223037337\
   6dba2+30870844232377/2230373376dba+24027/4425344db-7/256da5+5/96da3+34222\
   8711385/101380608da2+342235311685/25345152da+2645/2489256d-68441518085/57\
   931776b6+8617/1896576b5a-752908955671/318624768b5+7/256b4a2-2163/79024b4a\
   -752734601911/1115186688b4+529/316096b3a3+7/128b3a2-12811/59268b3a-752421\
   061495/557593344b3+311787/632192b2a4+529/158048b2a3+1/64b2a2-309/19756b2a\
   +309/4939b2+68441518085/22529024ba5+752978795959/61954816ba4+529/553168ba\
   3+1/32ba2-1/8ba+1/2b+3/32a6+68445742277/11264512a5+68447062337/2816128a4+\
   529/276584a3
↦  J[10]=cb-452831001/4597760d2ba2-3340189909/75863040d2ba-3176523/158048d2a\
   2-76214943/20230144d2a+5145/316096d2+25404981/101150720db4a-7203/1024db4+\
   453789/790240db3a2+125486265849/20230144db3a-25399101/12643840db3-324051/\
   6321920db2a3+194523/790240db2a2+11921259/25287680db2a-9921275/1264384db2-\
   27783/632192dba4-12813612147/50575360dba3-12806449187/75863040dba2+104952\
   523865/15172608dba-31286843/3160960db+1163396585/689664da2+1163396585/172\
   416da-172823/79024d-1628855795/2758656b6+64827/3160960b5a-89587760437/758\
   63040b5+64827/1580480b4a-12804376067/37931520b4+1323/1580480b3a3+9261/790\
   240b3a-12798251491/18965760b3-583443/395120b2a4-10887849/1580480b2a3+9261\
   /395120b2a-9261/98780b2+698081055/459776ba5+38375787849/6321920ba4-622295\
   1/395120ba3+698037951/229888a5+698037951/57472a4-1555407/395120a3
↦  J[11]=cd+66199/8d2ba2+6818809/2304d2ba+35/64d2b+47/96d2a2-155/768d2a-7/19\
   2db5+7/256db4a+7/320db4-1/80db3a2-133413949/256db3a-23/96db3+441/128db2a3\
   +1/64db2a+1/80db2+667316711/31360dba3+667069751/47040dba2-27349858321/470\
   40dba-1/8db+141/160da5+141/40da4+63/32da3-667069751/4704da2-667069751/117\
   6da+667069751/13440b6+667069751/6720b5-63/80b4a2+667069751/23520b4-63/40b\
   3a2+667069751/11760b3-9/20b2a2-2001209253/15680ba5-2001209253/3920ba4-9/1\
   0ba2-2001209253/7840a5-2001209253/1960a4
↦  J[12]=c2+1/3cb2+5/3d+3a3
intvec w = 3,2,1,2;
ring S3 = 0, (c,d,b,a), (a(w),lp);
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a (a(w),lp)-Groebner basis.
ideal J = modrWalk(I,radius,"Dp",w);
J;
↦  J[1]=d2
↦  J[2]=c2+3a3+1/3cb2+5/3d
↦  J[3]=ca2+4ca+7/2b3+2b
↦  J[4]=cda-6/7ba3+4/7c2-2/21cb3+1/7dba-10/21db
↦ J[5]=db4a-192/49b2a3+128/49c2b-64/7cda-64/147cb4+4/63d2ba+2db3a-4db4+60/4\
   9db2a-8db3+8/7dba-656/147db2-32/7db
↦ J[6]=db6+333576da3-189/4b8+15876b4a2-5186640/2401b2a3+44514592/2470629c2b\
   +772122956311640/155649627cd2+1815960/49cdb2-3535382608/352947cda+8395442\
   084/7203cb4-2304/49cb2a-826877376/2401ca2+308820941902337/311299254d2ba-4\
   3215/7db5+199982018/49db3a-4664160/7dba2-126b7+72/7b5a+31752b3a2+1728/240\
   1ba3-7028226304/2470629c2-1012272/49cdb+5598114168/2401cb3-4608/49cba-191\
   05043/21609d2b2-129552/49db4+864216/2401db2a-117b6+144/7b4a+9072b2a2-2016\
   0cd+4663152/7cb2-3307509504/2401ca-128772920/352947d2b-172796/49db3+57587\
   01763736/2470629dba-1296/7b5+288/49b3a+18144ba2+9326304/7cb+185320d2-1952\
   9120/7203db2-1188/7b4+576/49b2a+2752/2401db-413460864/343b3-3312/49b2-165\
   3754752/2401b
↦ J[7]=cb3a-5/8da3-245/1024b8-7/32b6a+2/343b2a3-1833054229/38118276c2b-3473\
   26678687775/9605805552cd2+5/112cdb2-916402091/5445468cda-108013/49392cb4+\
   25/7cb2a-966391/1372ca2-2222840154809345/307385777664d2ba-3/256db5-15431/\
   2016db3a+5/4dba2-245/256b7-25/32b5a+5/1372ba3-916476179/9529569c2+5/84cdb\
   -108035/24696cb3+16/7cba-3313805/790272d2b2-47/2688db4+53/24696db2a-91/25\
   6b6-5/8b4a-5/84cd-5/4cb2-966979/343ca-80483075/43563744d2b+1/192db3-21666\
   16927/76236552dba+65/32b5-1/14b3a-5/2cb-25/72d2-667/98784db2+85/64b4-2/7b\
   2a+109/12348db-1932803/784b3+3/14ba+93/112b2-966979/686b
↦ J[8]=cb5-4/7db2a2+96/7b2a3-64/7c2b-4/441cdb2+32cda+116/21cb4+1152/7ca2-8/\
   7dba2-8/441cdb+32/7cb3+14db4-16/7db2a+16/7cb2+4608/7ca+28db3+16/7cb+328/2\
   1db2+16db+576b3+2304/7b
↦ J[9]=cdb3+18da3+2cdb2+63cb4+441/2db3a-36dba2+126cb3+36cb2+126dba+72cb+10d\
   2
↦ J[10]=ba4-2/3c2a+1/9cb3a+14/3cda-1/6dba2+49/12db3+5/9dba+7/3db
↦ J[11]=b3a3+2744/5c2d+382/15c2b2-1127/15cdb3+693167/15cdba+129659/45cb5+23\
   52/5cb3a-8647/27d2b2a-49/15db4a-8232/5db2a2-1029/10b6a+616/15c2b+86470/27\
   cd2+161/5cdb2+1382932/15cda+28812/5cb4+1680cb2a+686/5d2ba-98/15db5-121045\
   /6db3a-735/2b5a-112/5c2-224/15cdb+8232/5cb3+5376/5cba+201698/5db4+38/3db2\
   a+2058/5b6-294b4a-28cd+16464/5cb2-4032/5ca+3631597/45db3-173024/15dba+147\
   0b5-168/5b3a+115256/5db2+1176b4-672/5b2a+138352/3db+672/5b3+504/5ba+2688/\
   5b2-2016/5b
↦ J[12]=b5a2+2420474/3c2d+32/9c2b2-1210217/9cdb3-689428/147cdba+931/2cb5+64\
   cb3a-345872/7cba2-6173/81d2b4-1867616522/3969d2b2a-7/18db6+669815/252db4a\
   -117310/441db2a2-14b6a+4b4a2+395552/63c2b+18676165220/3969cd2+194/63cdb2+\
   1844168/147cda-8469797cb4+1600/7cb2a-128/7ca2+7471160423/7938d2ba-11/6db5\
   -622555298/21db3a+2134623448/441dba2-50b5a+32/7b3a2+790208/63c2-124/21cdb\
   -16941190cb3-1382464/7cba-24692/567d2b2-259351/63db4+74519/49db2a+56b6-40\
   b4a+16/7b2a2-400/63cd-4839884cb2-1280/7ca-518474/63db3-7469280824/441dba+\
   200b5-32/7b3a+16/7ba2-9680832cb-1037348/441db2-172776b4-128/7b2a-2073632/\
   441db-320/7b3+96/7ba-691232/7b2-640/7b
↦ J[13]=b7a-13606662/245c2d+1024/735c2b2-864700/27783cd2b+23004293/2205cdb3\
   +29189312/12005cdba+758/5cb5+3904/245cb3a+12618cba2+12600644807/388962d2b\
   2a+4/63db6+537472/15435db4a-3032/35db2a2-52/35b6a-1008b4a2+1266752/15435c\
   2b-21001940345/64827cd2+20176/15435cdb2+554322208/108045cda+20430577/35cb\
   4+26496/343cb2a+9216ca2-2334414076/36015d2ba+863872/2205db5+12600769819/6\
   174db3a-16335524/49dba2-4b7-772/49b5a-2016b3a2+2447488/15435c2+2892256/22\
   05cdb+40842962/35cb3+12382792/245cba+33193268/15435db4+49664/2401db2a+208\
   /35b6-736/49b4a-576b2a2+3946912/3087cd+81722308/245cb2+63150592/1715ca+10\
   0/441d2b+9239464/2205db3+126015926254/108045dba+3088/49b5-432/1715b3a-115\
   2ba2+32671944/49cb+44256944/36015db2+2166931/49b4-1984/245b2a+16325536/72\
   03db+55320768/1715b3+8896/1715ba+6190756/245b2+31575296/1715b
```

```
↦ J[14]=b9-48392128/5145c2d+469223949568/2334744405c2b2+4446618588746560/29\
   417779503cd2b+8065888/5145cdb3+293190604544/333534915cdba+384/35cb5+3072/\
   1715cb3a-11648320/16807cba2+1050576605312618/29417779503d2b2a-4/441db6-15\
   04/108045db4a-1536/245db2a2+4b8-96/245b6a-256/16807b2a3+767602156544/2334\
   744405c2b-10670463200/194481cd2+7936/36015cdb2+23968768/252105cda+7472643\
   9712/756315cb4+16384/2401cb2a+2651044/151263d2b3-74705512256/6806835d2ba-\
   664/15435db5+7469324432/21609db3a-19361152/343dba2+36/7b7-512/343b5a-1757\
   184/12005c2+256/108045cdb+48404416/245cb3-232478976/84035cba+515091680/66\
   706983d2b2+16596304/108045db4+46922869120/466948881db2a+1504/245b6-512/34\
   3b4a-1024/21609cd+96827264/1715cb2-16384/12005ca-640/3087d2b+33201824/108\
   045db3+49786270144/252105dba+3728/343b5-6656/12005b3a+38722304/343cb+9483\
   008/108045db2-5806688/2401b4-8704/12005b2a+8854528/50421db+42304/12005b3+\
   2048/12005ba-116239488/84035b2-8192/12005b
↦ J[15]=dba3-2/3c2d+1/9cdb3-1/6d2ba+5/9d2b
↦ J[16]=db3a2-147/2cdb3-10cdba-288cba2+63b4a2-32c2b-132cda-576ca2-49/2db5+4\
   db3a+126b3a2-64c2-82cdb-1152cba-21/2db4+36b2a2-80cd-2304ca-14db3-16dba+72\
   ba2-6db2-1008b4-2016b3-576b2-1152b
↦ J[17]=a5+1/9cb2a2-4/3c2a-7/6cb3+5/9da2-2/3cb
↦ J[18]=da4-4/21c3+11/63cdba+8/63c2b-1/63db3a+10/63cdb+32/7cba+5/9d2a+2/63d\
   b2a-b4a+4b4-4/7b2a+16/7b2
```

See also: Section D.2.6.1 [modular], page 883.

D.15.10.3 modfWalk

Procedure from library `modwalk.lib` (see Section D.15.10 [modwalk_lib], page 2439).

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.

Example:

```
LIB "modwalk.lib";
ring R1 = 0, (x,y,z,t), dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
I = std(I);
ring R2 = 0, (x,y,z,t), lp;
ideal I = fetch(R1, I);
ideal J = modfWalk(I);
J;
↦ J[1]=z4+1/5z2+4/5
↦ J[2]=y5+1/11y3+2/11
↦ J[3]=x3+1/3x2+1/3
ring S1 = 0, (a,b,c,d), Dp;
ideal I = 5b2, ac2+9d3+3a2+5b, 2a2c+7abd+bcd+4a2, 2ad2+6b2d+7c3+8ad+4c;
I = std(I);
ring S2 = 0, (c,d,b,a), lp;
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a lp-Groebner basis.
ideal J = modfWalk(I,"Dp");
J;
↦ J[1]=a25+16a24+96a23+256a22+256a21+256/9a20+1024/3a19+2048a18+65536/9a17+\
   32768/3a16+16384/81a15+131072/81a14+1048576/81a13+1048576/27a12+1048576/9\
   a11
↦ J[2]=ba11+1522867351997104938459/91668001658017308797687087104a24+4293036\
   9782248629690765/91668001658017308797687087104a23+80925218629630777478637\
   /22917000414504327199421771776a22+7108535670237178684767/2864625051813040\
   899927721472a21-3255817194541612658349/89519532869157528122741296a20+5380\
   8965391546362724459/358078131476630112490965184a19+1534729815590907963215\
   01/358078131476630112490965184a18-260815719913165309506063/44759766434578\
   764061370648a17-1485276141860757031491027/89519532869157528122741296a16-4\
   92332725360316960775/22379883217289382030685324a15+7423992361030571232440\
   /16784912412967036523013993a14-17640364913371983121693/167849124129670365\
   23013993a13-37723213977586186442564/5594970804322345507671331a12+92047580\
   41857159721472414/5594970804322345507671331a11
↦ J[3]=b2a6-1275460856846934902527/2619112706933760ba13-9203207900045442436\
   039/4583447237134080ba12-86229451659721876411/6261539941440ba11-63/2ba10-\
   176103105371/907641397248a21-306836641361771/103471119286272a20-556810728\
   2265313/181074458750976a19+463053737062394039071/32593402575175680a18+233\
   493474751858876283/2037087660948480a17+66864105293320847653/2910125229926\
   40a16-2870620980805387/101854383047424a15-9651151354818523/25463595761856\
   a14-471264260028168789475/611126298284544a13-489585210177544137565/152781\
   574571136a12-4323891804194718805849/190976968213920a11+4/7a6
↦ J[4]=b3a5+1165198340059361507/60324030330315296b2a8+2834258341382482451/3\
   3932267060802354b2a7-3535202147650393/45243022747736472b2a6-1743263488518\
   66272086637/108100662351925010432ba13-180877903989284763838711/2702516558\
   7981252608ba12-2783949606995368438959121/60806622572957818368ba11+7423924\
   5100658253/30162015165157648ba10+4/7ba5+82576115266283735625891/172961059\
   7630800166912a18+2949131532722419989261/7721475882280357888a17+8257374787\
   8379091193883/108100662351925010432a16-194476798666289267/270251655879812\
   52608a15+10816123814459571297/6756291396995313152a14-13707944038216024518\
   9169/54050331175962505216a13-143935810252288492483523/1351258279399062630\
   4a12-108976370407053801898907/1447776727927567104a11+9/2a10+18a9+11651983\
   40059361507/105567053078051768a8+5668516682764964902/118762934712808239a7\
   -3535202147650393/79175289808538826a6
↦ J[5]=b4-63/2b3a4+53590985045967705613474269749/1407238979545595225088b2a8\
   +11625419776309240542435260807/58634957481066467712b2a7+12739505342603800\
   3554252983797/175904872443199403136b2a6+4/7b2-761164231440237277536357328\
   493047/700492292040474067599360ba13-104521491576210733019159653207691/250\
   17581858588359557120ba12-11826338279466524349922531923473911/394026914272\
   766663024640ba11+286749359828001390655523145503/39089971654044311808ba10-\
   61261515/8ba9-525086793/32ba8+3969/16ba7-19845/4ba6+9/2ba5+45295478141475\
   95860677850296017/138368847810464013352960a18+247345844501521328455296898\
   6663/9729059611673250938880a17+324025940290811430696988658749687/70049229\
   2040474067599360a16-4286410117811597590771586510563/350246146020237033799\
   68a15+4054710919870705842300540149/2918717883501975281664a14-394300742086\
   35887294045314648189/23349743068015802253312a13-1139723514755969617170188\
   24182045/17512307301011851689984a12-3226716701597602362857429388525683/65\
   671152378794443837440a11+771901337907/64a10-1701/4a9+80528689834593164587\
   767860213/2462668214204791643904a8+11625165236461787969048941667/10261117\
   5591866318496a7+127391998947868572673617154117/307833526775598955488a6
↦ J[6]=da+13042589597338317401292876694323783641/32769687632608435019205956\
   084751360b8-339347956624707128686162624470779591/136540365135868479246691\
   48368646400b7a+44776090034814931064318387941081165483/3276968763260843501\
   9205956084751360b7+72711086789965200830753309360861163/341350912839671198\
   1167287092161600b6a2+17945571338576304647577664032291913/1092322921086947\
    833973531869491712b6a+1005252013115630519751071456639664155699/5734695335\
    70647612836104231483148800b6+9615448034722990303485382435238901/136540365\
    13586847924669148368646400b5a3+663547630395349792235158387078347149/13654\
    036513586847924669148368646400b5a2-10588152408989285294663152330377654/37\
    3352560918390372940172025705175b5a+8954607782583032024373974051595847951/\
    5734695335706476128361042314831488b5-16768783292422657909931339309776827/\
    6827018256793423962334574184323200b4a4+507941899552434600769579126963767/\
    170675456419835599058364354608080b4a3+3300581091717697309395446699927846\
    9/6827018256793423962334574184323200b4a2+64107680452610290564228851734203\
    787/3413509128396711981167287092161600b4a+1376877864134098113390589464572\
    831613/853377282099177995291821773040400b4-643284910890587948236877864337\
    714771/9557825559510793547268403858052480b3a5-130190339942501750272181116\
    13508381/11947281949388491934085504822565600b3a4+137363543353185575764076\
    8919319843/3413509128396711981167287092161600b3a3+66354763039534979223515\
    8387078347149/23894563898776983868171009645131200b3a2-1380791010410641347\
    641955610911797/170675456419835599058364354608080b3a+30455774007493462026\
    8172466495764041/682701825679342396233457418432320b3+57088298465273022117\
    56158284190858113/95578255595107935472684038580524800b2a6+188328905241473\
    4766990421180196597/5973640974694245967042752411282800b2a5-20182594001309\
    76681444622383138147/11947281949388491934085504822565600b2a4+725631285074\
    90657252797018137681/42668864104958899764591088652020b2a3+352799606098503\
    36937362983980868571/1706754564198355990583643546080800b2a2+9160576385145\
    641871627802519827657/1706754564198355990583643546080800b2a+5063824837275\
    183915708779051981113317/11947281949388491934085504822565600b2+2139710638\
    608916733916466261891227/1365403651358684792466914836864640ba8-2207819314\
    4537153891260371881262339/546161460543473916986765934745856ba7-5451366058\
    15586548522667378314077621/3413509128396711981167287092161600ba6-65631655\
    441612907465368619382679107/1706754564198355990583643546080800ba5-189/200\
    ba4-13233263602544734530107233305922887/682701825679342396233457418432320\
    0a9-3555188132635967197804852869818541/1365403651358684792466914836864640\
    a8+230691784778909201021851072404959103/477891277975539677363420192902624\
    00a7-2239172271256758942944178285566542047/238945638987769838681710096451\
    31200a6+9/5a4
↦    J[7]=db2-52682490423871082296629464283815234/5824767849546495553439269268\
    4816735da+4/7d-37194377146998694799849872861154888794159/6390602440645297\
    9786305125688484646400b8+5562401664632673692771437240347563428391/9319628\
    5592743928855028308295706776000b7a-60969992378608072411136561729066173884\
    2371/319530122032264898931525628442423232000b7-57382636249315287334130146\
    46817239405443/124261714123658571806704411060942368000b6a2-10847266966441\
    319068560579895722895039907/372785142370975715420113233182827104000b6a-27\
    35710626254612029118700402548331033168619/1118355427112927146260339699548\
    481312000b6+180895821889433491876619053925823351/340070372533274690220865\
    93065392000b5a3-13485580971560996606666068965982819343311/186392571185487\
    857710056616591413552000b5a2+3705875600138678704564240261243242140767/543\
    64499929100625165433179839162286000b5a-6096801718704727096493321985237292\
    10634851/279588856778231786565084924887120328000b5+4111550062925787349119\
    083832946689053/1479306120519744902460766798344552000b4a4-329990252397055\
    256446695220829192741/147930612051974490246076679834455200b4a3-5500816426\
    513609089701808086792453204443/72485999905467500220577573118883048000b4a2\
    -10849280138379535415092613505685915291543/326186999574603750992599079034\
    973716000b4a-2469122007101831733334304136225298205357/1109479590389808676\
    845575098758414000b4+778576881370029841362906650943590232049/621308570618\
    2928590335220553047118400b3a5+16417021965805756241925867114270711739/1553\
    27142654573214758380513826177960000b3a4+12465668403174318998134335064393l3\
    31/5951231519332307078865153786443600b3a3-13485580971560996606666068965 9\
```

```
   271426545732147583805138261779600 0b3a4+1246566840317431899813433506439313\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/5951231519332307078865153786443600 0b3a3-1348558097156099660666606896 59\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/32618699957460375099259907903497371600 0b3a2+11337194154659923\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/58247678495464955534392692684816735 0b3a-20322013998\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/14497199981093500044115514623 7\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980 00b2a4+13599117556666438782844632064934113 9/25887857\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-10983862912866631158869805728122571 6407/\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   38831785663643303689595128456544490 00b2a2-3515158182804584308916309399303\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795 36\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/543644999291006251654331798391622860 00b2+4763451990675051\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   5463212340832732606276 37/48323999936978333480385048745922032000ba7+157734\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   939413816073987898814416077156887 9/51775714218191071586126837942059320 00b\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+129806152916316824908081593163418714 6059/18121499976366875055144393279\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+390038731018679326132909902806933519 1/17258571406\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+3900387310186793261329099028069335191/17258571406\
   0636905287089459806864400 0a9-311663413071729413785792362293463537 9/493102\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+3900387310186793261329099028069335191/17258571406\
   06369052870894598068644000a9-3116634130717294137857923622934635379/493102\
   0401732483008202555994481840 00a8-2437307904588001673427936587902645802933\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+3900387310186793261329099028069335191/17258571406\
   06369052870894598068644000a9-3116634130717294137857923622934635379/493102\
   04017324830082025559944818400 0a8-2437307904588001673427936587902645802933\
   /7248599990546750022057757311888304800 0a7+4675971994304336297474732550069\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+3900387310186793261329099028069335191/17258571406\
   06369052870894598068644000a9-3116634130717294137857923622934635379/493102\
   040173248300820255599448184000a8-2437307904588001673427936587902645802933\
   /72485999905467500220577573118883048000a7+4675971994304336297474732550069\
   231267517/362429999527337501102887865594415240 00a6-526824904238710822966 2\
```

```
   2714265457321475838051382617796000b3a4+1246566840317431899813433506439313\
   31/59512315193323070788651537864436000b3a3-134855809715609966066660689659\
   82819343311/326186999574603750992599079034973716000b3a2+11337194154659923\
   308002619476290128559/582476784954649555343926926848167350b3a-20322013998\
   4954898395766259918932227475777/326186999574603750992599079034973716000b3\
   -17539768590087896897467462966094000442659/144971999810935000441155146237\
   766096000b2a6-12888123469427488410840380363220212653/27182249964550312582\
   716589919581143000b2a5+83962014758738499482259965901772639/77663571327286\
   607379190256913088980000b2a4+135991175566664387828446320649341139/25887857\
   1090955357930634189710296600b2a3-109838629128666311588698057281225716407/\
   3883178566364330368959512845654449000b2a2-3515158182804584308916309399303\
   186117/369826530129936225615191699586138000b2a-316153073927461285155795536\
   862930855381313/54364499929100625165433179839162286000b2+4763451990675051\
   546321234083273260627637/48323999936978333480385048745922032000ba7+157734\
   9394138160739878988144160771568879/5177571421819107158612683794205932000b\
   a6+1298061529163168249080815931634187146059/18121499976366875055144393279\
   720762000ba5+70513412412278721745933349799575459/616377550216560376025319\
   49931023000ba4-27/50ba3+3900387310186793261329099028069335191/17258571406\
   06369052870894598068644000a9-3116634130717294137857923622934635379/493102\
   040173248300820255599448184000a8-2437307904588001673427936587902645802933\
   /72485999905467500220577573118883048000a7+4675971994304336297474732550069\
   231267517/36242999952733750110288786559441524000a6-5268249042387108229662\
   9464283815234/32359821386369419741329273713787075a4+36/35a3
↦ J[8]=d2
↦ J[9]=ca-1630191535/8275968d2ba2-10498813705/148967424d2ba-456425375/60690\
   432d2a+92575/2844864d2+91285075/182071296db4a+35/9216db4-1088785/5689728d\
   b3a2+2258752616171/182071296db3a+24027/2528768db3-8162999/79656192db2a3-6\
   539063/79656192db2a2+25495277/45517824db2a+5095/1264384db2-5243/158048dba\
   4-753114969703/1486915584dba3-753467060575/2230373376dba2+30870844232377/\
   2230373376dba+24027/4425344db-7/256da5+5/96da3+342228711385/101380608da2+\
   342235311685/25345152da+2645/2489256d-68441518085/57931776b6+8617/1896576\
   b5a-752908955671/318624768b5+7/256b4a2-2163/79024b4a-752734601911/1115186\
   688b4+529/316096b3a3+7/128b3a2-12811/59268b3a-752421061495/557593344b3+31\
   1787/632192b2a4+529/158048b2a3+1/64b2a2-309/19756b2a+309/4939b2+684415180\
   85/22529024ba5+752978795959/61954816ba4+529/553168ba3+1/32ba2-1/8ba+1/2b+\
   3/32a6+68445742277/11264512a5+68447062337/2816128a4+529/276584a3
↦ J[10]=cb-452831001/4597760d2ba2-972112421/27586560d2ba-76214943/20230144d\
   2a+5145/316096d2+25404981/101150720db4a-7203/1024db4+453789/790240db3a2+1\
   25486265849/20230144db3a-25399101/12643840db3-324051/6321920db2a3+194523/\
   790240db2a2+11921259/25287680db2a-9921275/1264384db2-27783/632192dba4-128\
   13612147/50575360dba3-12806449187/75863040dba2+104952523865/15172608dba-3\
   1286843/3160960db+1163396585/689664da2+1163396585/172416da-172823/79024d-\
   1628855795/2758656b6+64827/3160960b5a-89587760437/75863040b5+64827/158048\
   0b4a-12804376067/37931520b4+1323/1580480b3a3+9261/790240b3a-12798251491/1\
   8965760b3-583443/395120b2a4-10887849/1580480b2a3+9261/395120b2a-9261/9878\
   0b2+698081055/459776ba5+38375787849/6321920ba4-6222951/395120ba3+69803795\
   1/229888a5+698037951/57472a4-1555407/395120a3
↦ J[11]=cd+1148200/372848203cb-1524024/27175525ca+5764250385873/695693440d2\
   ba2+4124562494101/1391386880d2ba+35/64d2b+5/96d2a2+5/24d2a-7717/4348084d2\
   -7/192db5-6353027389387677/12174635200db3a-4497962717/18261952800db3+4524\
   5274457/13111145600db2a3+228471973/42611223200db2a2-21815387/1521829400db\
   2a-516999/43480840db2+750069/434808400dba4+907911207105139/42611223200dba\
   3+30252540938799/2130561160dba2-1771932852381621/3043658800dba-237078827/\
   1521829400db+383841663/434808400da5+141/40da4+341903607/173923360da3-6353\
   027418401357/44741784360da2-12706055163515359/22370892180da-73857/1087021\
   0d+1728714797635/34784672b6-5208/27175525b5a+6353027467398221/63916834800\
   b5-686156751/869616800b4a2+27783/16723400b4a+6173979975409/217404200b4-69\
   453/760914700b3a3-686156751/434808400b3a2+1321617/108702100b3a+1270604395\
   725157/22370892180b3-14003199/434808400b2a4-4652991/217404200b2a3-9802239\
   3/217404200b2a2+3969/4180850b2a-7938/2090425b2-2222633311245/17392336ba5-\
   55565845783569/108702100ba4-9236529/190228675ba3-98022393/108702100ba2+19\
   0503/27175525ba-762012/27175525b-571509/108702100a6-19059082255204071/745\
   69640600a5-38118165490546077/37284820300a4-664713/54351050a3
↦ J[12]=c2+1/3cb2+5/3d+3a3
intvec w = 3,2,1,2;
ring S3 = 0, (c,d,b,a), (a(w),lp);
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a (a(w),lp)-Groebner basis.
ideal J = modfWalk(I,"Dp",w);
J;
↦ J[1]=d2
↦ J[2]=c2+3a3+1/3cb2+5/3d
↦ J[3]=ca2+4ca+7/2b3+2b
↦ J[4]=cda-6/7ba3-2/21cb3-12/7a3-4/21cb2+1/7dba-10/21db-20/21d
↦ J[5]=db4a-192/49b2a3-64/147cb4+2db3a-768/49ba3-256/147cb3-4db4+60/49db2a-\
   768/49a3-256/147cb2-8db3+120/49dba-656/147db2-1952/147db-1280/147d
↦ J[6]=db6+333576da3-189/4b8+15876b4a2-5186640/2401b2a3+1815960/49cdb2+8395\
   442084/7203cb4-2304/49cb2a-43215/7db5+199982018/49db3a-4664160/7dba2-126b\
   7+72/7b5a+31752b3a2-20742528/2401ba3-1012272/49cdb+16787427752/7203cb3-46\
   08/49cba-129552/49db4+864216/2401db2a-117b6+144/7b4a+9072b2a2-20738496/24\
   01a3-20160cd+4791470576/7203cb2-172796/49db3+5599841736/2401dba-1296/7b5+\
   288/49b3a+18144ba2+9326304/7cb-19529120/7203db2-1188/7b4+576/49b2a-345655\
   04/7203db-3168/49b3-34564160/7203d-3312/49b2
↦ J[7]=cb3a-5/8da3-245/1024b8-7/32b6a+2/343b2a3+5/112cdb2-108013/49392cb4+2\
   5/7cb2a-3/256db5-15431/2016db3a+5/4dba2-245/256b7-25/32b5a+8/343ba3+5/84c\
   db-107981/24696cb3+16/7cba-47/2688db4+53/24696db2a-91/256b6-5/8b4a+8/343a\
   3-5/84cd-15403/12348cb2-12/7ca+1/192db3-12015/2744dba+65/32b5-1/14b3a-5/2\
   cb-667/98784db2+85/64b4-2/7b2a+61/3087db-3/112b3+3/14ba+40/3087d+93/112b2\
   -6/7b
↦ J[8]=cb5-4/7db2a2+96/7b2a3-4/441cdb2+116/21cb4-8/7dba2+384/7ba3-8/441cdb+\
   32/3cb3+14db4-16/7db2a+384/7a3+176/21cb2+28db3-32/7dba+16/7cb+328/21db2+9\
   76/21db+640/21d
↦ J[9]=cdb3+18da3+2cdb2+63cb4+441/2db3a-36dba2+126cb3+36cb2+126dba+72cb
↦ J[10]=ba4+5/72da3+245/9216b8+7/288b6a-2/3087b2a3+2a4-5/1008cdb2+108013/44\
   4528cb4-11/63cb2a+1/768db5+15431/18144db3a-11/36dba2+245/2304b7+25/288b5a\
   +12340/3087ba3-5/756cdb+206765/222264cb3-16/63cba+47/24192db4-53/222264db\
   2a+91/2304b6+5/72b4a+24688/3087a3+5/756cd+114187/111132cb2+4/21ca+7055/17\
   28db3+9271/24696dba-65/288b5+1/126b3a+5/18cb+667/889056db2+10/9da-85/576b\
   4+2/63b2a+126506/27783db+1/336b3-1/42ba+123440/27783d-31/336b2+2/21b
↦ J[11]=b3a3+4/63db2a2+7203/64b8+52/21b2a3+18526/3969cdb2-32/189cb4-49/48db\
   5-1/6db3a+8/63dba2+7203/16b7-44/21ba3+37052/3969cdb-20/27cb3-259/72db4-5/\
   63db2a+9261/16b6-128/21a3-176/189cb2-113/36db3+32/63dba+1029/2b5-16/63cb-\
   257/378db2+2205/4b4-556/189db+147b3-640/189d+147b2
↦ J[12]=b5a2-4/441db2a2+129724da3-49/16b8+6178b4a2-864464/1029b2a3+302660/2\
   1cdb2+4197721018/9261cb4-128/7cb2a-28823/12db5+99991009/63db3a-114271928/\
   441dba2+49/4b7+4b5a+86468/7b3a2-3457184/1029ba3-506216/63cdb+8393713780/9\
   261cb3-256/7cba-129625/126db4+432008/3087db2a+133/4b6+8b4a+24712/7b2a2-34\
   56512/1029a3-494080/63cd+2395735192/9261cb2-86405/63db3+2799920668/3087db\
   a-2b5+16/7b3a+49408/7ba2+518128cb-9766570/9261db2+9b4+32/7b2a-17280544/92\
   61db-36/7b3-17282560/9261d-44/7b2
↦ J[13]=b7a-296432/7da3+109/16b8+2b6a-2016b4a2+4610560/16807b2a3-14521796/3\
   087cdb2-22401156488/151263cb4+8000/343cb2a+1382867/1764db5-1600806884/308\
   7db3a+4148384/49dba2+77/4b7-194/49b5a-4032b3a2+18438656/16807ba3+8103808/\
   3087cdb-44793095440/151263cb3+13568/343cba+230189/686db4-6908840/151263db\
   2a+533/28b6-312/49b4a-1152b2a2+18435072/16807a3+7901200/3087cd-4261620128\
   /50421cb2-4864/343ca+1382435/3087db3-4980595264/16807dba+2078/49b5+176/34\
   3b3a-2304ba2-8294976/49cb+52056614/151263db2+2435/49b4-1472/343b2a+921811\
   84/151263db+1900/343b3+608/343ba+30725120/50421d+8044/343b2-2432/343b
↦ J[14]=b9+1037536/343da3+4b8+144b4a2-329408/16807b2a3+148256/441cdb2+15987\
   36592/151263cb4-24704/441db5+799730504/21609db3a-296064/49dba2+40/7b7+288\
   b3a2-1317376/16807ba3-4047040/21609cdb+3196814624/151263cb3+2048/2401cba-\
   24db4+164704/50421db2a+40/7b6-64/343b4a+576/7b2a2-3840/49a3-3950080/21609\
   cd+886720/147cb2+4096/2401ca-98816/3087db3+1036320/49dba+272/49b5-128/343\
   b3a+1152/7ba2+592000/49cb-3721504/151263db2+1152/343b4-256/2401b2a-658688\
   0/151263db+1024/343b3-512/2401ba-6400/147d+1920/2401b2+2048/2401b
↦ J[15]=dba3-7cb4-49/2db3a+4dba2-14cb3-4cb2-14dba-8cb
↦ J[16]=db3a2+1323da3+63b4a2-60/7b2a3+147cdb2+194441/42cb4-49/2db5+64843/4d\
   b3a-2646dba2+126b3a2-240/7ba3-82cdb+194401/21cb3-21/2db4+10/7db2a+36b2a2-\
   240/7a3-80cd+55486/21cb2-14db3+64847/7dba+72ba2+5292cb-226/21db2-400/21db\
   -400/21d
↦ J[17]=a5+4a4-7/6cb3+5/9da2-7/18b5-2/3cb+20/9da-2/9b3
↦ J[18]=da4-2/49b2a3-2/441cb4-1/63db3a-4/49ba3+10/63cdb-4/441cb3+32/7cba+1/\
   147db2a-b4a+20/63cd+64/7ca-2b3a-10/441db2+4b4-4/7b2a-20/441db+8b3-8/7ba+1\
   6/7b2+32/7b
```

See also: Section D.2.6.1 [modular], page 883.

D.15.10.4 modfrWalk

Procedure from library `modwalk.lib` (see Section D.15.10 [modwalk_lib], page 2439).

Return: a standard basis of I

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods.

Example:

```
   LIB "modwalk.lib";
   ring R1 = 0, (x,y,z,t), dp;
   ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
   I = std(I);
   ring R2 = 0, (x,y,z,t), lp;
   ideal I = fetch(R1, I);
   int radius = 2;
   ideal J = modfrWalk(I,radius);
   J;
   ↦ J[1]=z4+1/5z2+4/5
   ↦ J[2]=y5+1/11y3+2/11
   ↦ J[3]=x3+1/3x2+1/3
   ring S1 = 0, (a,b,c,d), Dp;
```

```
ideal I = 5b2, ac2+9d3+3a2+5b, 2a2c+7abd+bcd+4a2, 2ad2+6b2d+7c3+8ad+4c;
I = std(I);
ring S2 = 0, (c,d,b,a), lp;
ideal I = fetch(S1,I);
// I is assumed to be a Dp-Groebner basis.
// We compute a lp-Groebner basis.
ideal J = modfrWalk(I,radius,"Dp");
J;
↦ J[1]=a25+16a24+96a23+256a22+256a21+256/9a20+1024/3a19+2048a18+65536/9a17+\
   32768/3a16+16384/81a15+131072/81a14+1048576/81a13+1048576/27a12+1048576/9\
   a11
↦ J[2]=ba11+1522867351997104938459/91668001658017308797687087104a24+4293036\
   9782248629690765/91668001658017308797687087104a23+80925218629630777478637\
   /22917000414504327199421771776a22+7108535670237178684767/2864625051813040\
   899927721472a21-3255817194541612658349/89519532869157528122741296a20+5380\
   8965391546362724459/358078131476630112490965184a19+1534729815590907963215\
   01/358078131476630112490965184a18-260815719913165309506063/44759766434578\
   764061370648a17-1485276141860757031491027/89519532869157528122741296a16-4\
   92332725360316960775/22379883217289382030685324a15+7423992361030571232440\
   /16784912412967036523013993a14-17640364913371983121693/167849124129670365\
   23013993a13-37723213977586186442564/5594970804322345507671331a12+92047580\
   41857159721472414/5594970804322345507671331a11
↦ J[3]=b2a6-1275460856846934902527/2619112706933760ba13-9203207900045442436\
   039/4583447237134080ba12-86229451659721876411/6261539941440ba11-63/2ba10-\
   176103105371/907641397248a21-306836641361771/103471119286272a20-556810728\
   2265313/181074458750976a19+463053737062394039071/32593402575175680a18+233\
   493474751858876283/2037087660948480a17+66864105293320847653/2910125229926\
   40a16-2870620980805387/101854383047424a15-9651151354818523/25463595761856\
   a14-471264260028168789475/611126298284544a13-489585210177544137565/152781\
   574571136a12-4323891804194718805849/190976968213920a11+4/7a6
↦ J[4]=b3a5+1165198340059361507/60324030330315296b2a8+2834258341382482451/3\
   3932267060802354b2a7-3535202147650393/45243022747736472b2a6-1743263488518\
   66272086637/108100662351925010432ba13-180877903989284763838711/2702516558\
   7981252608ba12-2783949606995368438959121/60806622572957818368ba11+7423924\
   5100658253/30162015165157648ba10+4/7ba5+82576115266283735625891/172961059\
   7630800166912a18+2949131532722419989261/7721475882280357888a17+8257374787\
   8379091193883/108100662351925010432a16-194476798666289267/270251655879812\
   52608a15+10816123814459571297/6756291396995313152a14-13707944038216024518\
   9169/54050331175962505216a13-143935810252288492483523/1351258279399062630\
   4a12-108976370407053801898907/1447776727927567104a11+9/2a10+18a9+11651983\
   40059361507/105567053078051768a8+5668516682764964902/118762934712808239a7\
   -3535202147650393/79175289808538826a6
↦ J[5]=b4-63/2b3a4+53590985045967705613474269749/1407238979545595225088b2a8\
   +11625419776309240542435260807/58634957481066467712b2a7+12739505342603800\
   3554252983797/175904872443199403136b2a6+4/7b2-761164231440237277536357328\
   493047/700492292040474067599360ba13-104521491576210733019159653207691/250\
   17581858588359557120ba12-11826338279466524349922531923473911/394026914272\
   766663024640ba11+286749359828001390655523145503/39089971654044311808ba10-\
   61261515/8ba9-525086793/32ba8+3969/16ba7-19845/4ba6+9/2ba5+45295478141475\
   95860677850296017/138368847810464013352960a18+247345844501521328455296898\
   6663/9729059611673250938880a17+324025940290811430696988658749687/70049229\
   2040474067599360a16-4286410117811597590771586510563/350246146020237033799\
   68a15+4054710919870705842300540149/2918717883501975281664a14-394300742086\
   35887294045314648189/23349743068015802253312a13-1139723514755969617170188\
   24182045/17512307301011851689984a12-3226716701597602362857429388525683/65\
   671152378794443837440a11+771901337907/64a10-1701/4a9+80528689834593164587\
   767860213/2462668214204791643904a8+11625165236461787969048941667/10261117\
   5591866318496a7+127391998947868572673617154117/307833526775598955488a6
↦  J[6]=da+13042589597338317401292876694323783641/32769687632608435019205956\
   084751360b8-339347956624707128686162624470779591/136540365135868479246691\
   48368646400b7a+44776090034814931064318387941081165483/3276968763260843501\
   9205956084751360b7+72711086789965200830753309360861163/341350912839671198\
   1167287092161600b6a2+17945571338576304647577664032291913/1092322921086947\
   833973531869491712b6a+1005252013115630519751071456639664155699/5734695335\
   70647612836104231483148800b6+9615448034722990303485382435238901/136540365\
   13586847924669148368646400b5a3+663547630395349792235158387078347149/13654\
   036513586847924669148368646400b5a2-10588152408989285294663152330377654/37\
   3352560918390372940172025705175b5a+8954607782583032024373974051595847951/\
   5734695335706476128361042314831488b5-16768783292422657909931339309776827/\
   6827018256793423962334574184323200b4a4+507941899552434600769579126963767/\
   170675456419835599058364354608080b4a3+33005810917176973093954466999277846\
   9/6827018256793423962334574184323200b4a2+64107680452610290564228851734203\
   787/3413509128396711981167287092161600b4a+1376877864134098113390589464572\
   831613/853377282099177995291821773040400b4-643284910890587948236877864337\
   714771/9557825559510793547268403858052480b3a5-130190339942501750272181116\
   13508381/11947281949388491934085504822565600b3a4+137363543353185575764076\
   8919319843/3413509128396711981167287092161600b3a3+66354763039534979223515\
   8387078347149/23894563898776983868171009645131200b3a2-1380791010410641347\
   641955610911797/170675456419835599058364354608080b3a+30455774007493462026\
   8172466495764041/682701825679342396233457418432320b3+57088298465273022117\
   56158284190858113/95578255595107935472684038580524800b2a6+188328905241473\
   4766990421180196597/5973640974694245967042752411282800b2a5-20182594001309\
   76681444622383138147/11947281949388491934085504822565600b2a4+725631285074\
   90657252797018137681/42668864104958899764591088652020b2a3+352799606098503\
   36937362983980868571/1706754564198355990583643546080800b2a2+9160576385145\
   641871627802519827657/1706754564198355990583643546080800b2a+5063824837275\
   183915708779051981113317/11947281949388491934085504822565600b2+2139710638\
   608916733916466261891227/1365403651358684792466914836864640ba8-2207819314\
   4537153891260371881262339/546161460543473916986765934745856ba7-5451366058\
   15586548522667378314077621/3413509128396711981167287092161600ba6-65631655\
   441612907465368619382679107/1706754564198355990583643546080800ba5-189/200\
   ba4-13233263602544734530107233305922887/682701825679342396233457418432320\
   0a9-3555188132635967197804852869818541/1365403651358684792466914836864640\
   a8+230691784778909201021851072404959103/477891277975539677363420192902624\
   00a7-2239172271256758942944178285566542047/238945638987769838681710096451\
   31200a6+9/5a4
↦  J[7]=db2+27168629881138998278214605101276770835/1800336095813001708473756\
   2811777413434dba+491471906637994374111562338884716187596/4500840239532504\
   2711843907029443533585da2+2869014504205069356631324068220712761/300056015\
   9688336180789593801962902239da+4/7d-1897077401044693056678038097144471820\
   99/34292116110723842066166786308147454160b7a-5658327819535712436466574993\
   330286600457/122471843252585150236309951100526622000b7+113182841765429901\
   32449993783459998727469/2571908708304288154962508973111059062000b6a2-2253\
   0934741604865857767754519577491811561/17146058055361921033083393154073727\
   08000b6a-28490909495949003753054905391701315934163/5143817416608576309925\
   01794622211812400b6+163276752511472685450681124222320969/4898873730103406\
   0094523980440210648800b5a3+1147567905764537825599187468312120724677/12859\
   5435415214407748125448655552953100b5a2-4912488558205679461828306619738387\
   29409/60011203193766723615791876039258044780b5a-2432707710215176002654632\
   048907145876763/45926941219719431338616231662697483250b5-3344421008663233\
   279970959194674706231/12247184325258515023630995110052662200b4a4-14300645\
   742587052800896932744188700663/6123592162629257511815497555026331100b4a3+\
   18686441795456071650849593173135040994539/4500840239532504271184390702944\
   353358500b4a2-22563973604244056496552722465133853468563/15002800798441680\
   90394796900981451119500b4a-28424924011974492068513527513639082672303/4500\
   84023953250427118439070294435335850b4-16868448210344714519204977303940182\
   9737/12247184325258515023630995110052662200b3a5-8054343150103979756088591\
   5156427564921/85730290276809605165416965770368635400b3a4-2021296188976840\
   1048967087882194009871/21432572569202401291354241442592158850b3a3+1147567\
   905764537825599187468312120724677/225042011976625213559219535147217667925\
   b3a2-8615460449031389729000767207411187066/300056015968833618078959380196\
   2902239b3a-17082924484405326727765123704709182473311/11252100598831260677\
   96097675736088339625b3+58378083696210014254623732708487441906557/24004481\
   27750668944631675041570321791200b2a6+730752004376098538134949140639236486\
   3/75014003992208404519739845049072555975b2a5-1422484057185129510002160996\
   280908909/8573029027680960516541696577036863540b2a4+498866956969510836132\
   1884554144242302/10716286284601200645677120721296079425b2a3+1432100606202\
   736932633547014514099566/1530898040657314377953874388756582775b2a2-922327\
   03946462233205459960859960061737/21432572569202401291354241442592158850b2\
   a-1350425644190475256379626173122707114783/750140039922084045197398450490\
   72555975b2-32293044481548109001440278438930613005987/30005601596883361807\
   89593801962902239000ba7+32682158823966231260410559538958529214519/3000560\
   159688336180789593801962902239000ba6-570641550756452819672611948091956948\
   9863/750140039922084045197398450490725559750ba5+3153694723460303096873813\
   53216972006239/150028007984416809039479690098145111950ba4-27/50ba3-156464\
   120892409756516976517553184828379/85730290276809605165416965770368635400a\
   8-4055993575064692139179730619963358700739/150028007984416809039479690098\
   145111950a7-3837465521578008641979577565856688937979/12002240638753344723\
   1583752078516089560a6+1474415719913983122334687016654148562788/7501400399\
   2208404519739845049072555975a5+25821130537845624209681916613986414849/150\
   02800798441680903947969009814511195a4+36/35a3
↦ J[8]=d2
↦ J[9]=ca-1630191535/8275968d2ba2-10498813705/148967424d2ba-456425375/60690\
   432d2a+92575/2844864d2+91285075/182071296db4a+35/9216db4-1088785/5689728d\
   b3a2-8514901559/16551936db3a+24027/2528768db3-8162999/79656192db2a3-65390\
   63/79656192db2a2+25495277/45517824db2a+5095/1264384db2-5243/158048dba4+10\
   341291579/495638528dba3+3407975985/247819264dba2-1278848389663/2230373376\
   dba+24027/4425344db-7/256da5+5/96da3-52059465655/371728896da2-17345088185\
   /30977408da+2645/2489256d+3475793945/70805504b6+8617/1896576b5a+346998764\
   1/35402752b5+7/256b4a2-2163/79024b4a+317214571/11264512b4+529/316096b3a3+\
   7/128b3a2-12811/59268b3a+3524198105/61954816b3+311787/632192b2a4+529/1580\
   48b2a3+1/64b2a2-309/19756b2a+309/4939b2-31282145505/247819264ba5-31160048\
   481/61954816ba4+529/553168ba3+1/32ba2-1/8ba+1/2b+3/32a6-31235679393/12390\
   9632a5-31221158733/30977408a4+529/276584a3
↦ J[10]=cb-32663547/65203625ca+155649627/834606400d2ba2+111379989/166921280\
   0d2ba-1029/41730320d2-23489436789/3338425600db4+1118544609/1669212800db3a\
   2-8739975699/2086516000db3a-33610801623/16692128000db3+324051/4173032000d\
   b2a3+4795316673/16692128000db2a2+398184633/2086516000db2a-1637653157/2086\
   51600db2-228089169/8346064000dba4+64354689/1043258000dba3+12075959/562656\
     00dba2-100059777713/12519096000dba-41315573089/4173032000db+228644829/166\
     92128000da5-10887849/417303200da3-650211767/312977400da2-10566706007/1251\
     909600da-456425543/208651600d+1637849941/2503819200b6+76085289/4173032000\
     b5a+2161613251/1564887000b5-228644829/16692128000b4a2+228385521/417303200\
     0b4a+121184063/782443500b4-1323/1043258000b3a3-228644829/8346064000b3a2+2\
     50383483/2086516000b3a+109854617/312977400b3-14385954051/8346064000b2a4-2\
     8754809209/4173032000b2a3-32663547/4173032000b2a2+32626503/1043258000b2a-\
     32626503/260814500b2-701935689/417303200ba5-2782574067/260814500ba4-10269\
     57141/65203625ba3-32663547/2086516000ba2+32663547/521629000ba-32663547/13\
     0407250b-97990641/2086516000a6-1950635301/521629000a5-31700118021/2086516\
     000a4-4107829887/1043258000a3
  ↦ J[11]=cd+1148200/372848203cb-1524024/27175525ca+5764250385873/695693440d2\
     ba2+4124562494101/1391386880d2ba+35/64d2b+5/96d2a2+5/24d2a-7717/4348084d2\
     -7/192db5-325268570963421/3746041600db3a-4497962717/18261952800db3+452452\
     74457/13111145600db2a3+228471973/42611223200db2a2-21815387/1521829400db2a\
     -516999/43480840db2+750069/434808400dba4+605413666273951/170444892800dba3\
     +8054316673457/3408897856dba2-1179375331049779/12174635200dba-237078827/1\
     521829400db+383841663/434808400da5+141/40da4+341903607/173923360da3-42284\
     91538579193/178967137440da2-325268630154191/3441675720da-73857/10870210d+\
     442542060723/53514880b6-5208/27175525b5a+4228491734566649/255667339200b5-\
     686156751/869616800b4a2+27783/16723400b4a+316101027417/66893600b4-69453/7\
     60914700b3a3-686156751/434808400b3a2+1321617/108702100b3a+845693955895381\
     /89483568720b3-14003199/434808400b2a4-4652991/217404200b2a3-98022393/2174\
     04200b2a2+3969/4180850b2a-7938/2090425b2-568982649501/26757440ba5-2844917\
     248257/33446800ba4-9236529/190228675ba3-98022393/108702100ba2+190503/2717\
     5525ba-762012/27175525b-571509/108702100a6-12685474615737579/298278562400\
     a5-975805890462573/5736126200a4-664713/54351050a3
  ↦ J[12]=c2+1/3cb2+5/3d+3a3
  intvec w = 3,2,1,2;
  ring S3 = 0, (c,d,b,a), (a(w),lp);
  ideal I = fetch(S1,I);
  // I is assumed to be a Dp-Groebner basis.
  // We compute a (a(w),lp)-Groebner basis.
  ideal J = modfrWalk(I,radius,"Dp",w);
  J;
  ↦ J[1]=d2
  ↦ J[2]=c2+3a3+1/3cb2+5/3d
  ↦ J[3]=ca2+4ca+7/2b3+2b
  ↦ J[4]=cda-6/7ba3-2/21cb3-12/7a3-4/21cb2+1/7dba-10/21db-20/21d
  ↦ J[5]=db4a-192/49b2a3-64/147cb4+2db3a-768/49ba3-256/147cb3-4db4+60/49db2a-\
     768/49a3-256/147cb2-8db3+120/49dba-656/147db2-1952/147db-1280/147d
  ↦ J[6]=db6+333576da3-189/4b8+15876b4a2-5186640/2401b2a3+1815960/49cdb2+8395\
     442084/7203cb4-2304/49cb2a-43215/7db5+199982018/49db3a-4664160/7dba2-126b\
     7+72/7b5a+31752b3a2-20742528/2401ba3-1012272/49cdb+16787427752/7203cb3-46\
     08/49cba-129552/49db4+864216/2401db2a-117b6+144/7b4a+9072b2a2-20738496/24\
     01a3-20160cd+4791470576/7203cb2-172796/49db3+5599841736/2401dba-1296/7b5+\
     288/49b3a+18144ba2+9326304/7cb-19529120/7203db2-1188/7b4+576/49b2a-345655\
     04/7203db-3168/49b3-34564160/7203d-3312/49b2
  ↦ J[7]=cb3a-5/8da3-245/1024b8-7/32b6a+2/343b2a3+5/112cdb2-108013/49392cb4+2\
     5/7cb2a-3/256db5-15431/2016db3a+5/4dba2-245/256b7-25/32b5a+8/343ba3+5/84c\
     db-107981/24696cb3+16/7cba-47/2688db4+53/24696db2a-91/256b6-5/8b4a+8/343a\
     3-5/84cd-15403/12348cb2-12/7ca+1/192db3-12015/2744dba+65/32b5-1/14b3a-5/2\
     cb-667/98784db2+85/64b4-2/7b2a+61/3087db-3/112b3+3/14ba+40/3087d+93/112b2\
     -6/7b
  ↦ J[8]=cb5-4/7db2a2+96/7b2a3-4/441cdb2+116/21cb4-8/7dba2+384/7ba3-8/441cdb+\
     32/3cb3+14db4-16/7db2a+384/7a3+176/21cb2+28db3-32/7dba+16/7cb+328/21db2+9\
     76/21db+640/21d
  ↦ J[9]=cdb3+18da3+2cdb2+63cb4+441/2db3a-36dba2+126cb3+36cb2+126dba+72cb
  ↦ J[10]=ba4+5/72da3+245/9216b8+7/288b6a-2/3087b2a3+2a4-5/1008cdb2+108013/44\
     4528cb4-11/63cb2a+1/768db5+15431/18144db3a-11/36dba2+245/2304b7+25/288b5a\
     +12340/3087ba3-5/756cdb+206765/222264cb3-16/63cba+47/24192db4-53/222264db\
     2a+91/2304b6+5/72b4a+24688/3087a3+5/756cd+114187/111132cb2+4/21ca+7055/17\
     28db3+9271/24696dba-65/288b5+1/126b3a+5/18cb+667/889056db2+10/9da-85/576b\
     4+2/63b2a+126506/27783db+1/336b3-1/42ba+123440/27783d-31/336b2+2/21b
  ↦ J[11]=b3a3+4/63db2a2+7203/64b8+52/21b2a3+18526/3969cdb2-32/189cb4-49/48db\
     5-1/6db3a+8/63dba2+7203/16b7-44/21ba3+37052/3969cdb-20/27cb3-259/72db4-5/\
     63db2a+9261/16b6-128/21a3-176/189cb2-113/36db3+32/63dba+1029/2b5-16/63cb-\
     257/378db2+2205/4b4-556/189db+147b3-640/189d+147b2
  ↦ J[12]=b5a2-4/441db2a2+129724da3-49/16b8+6178b4a2-864464/1029b2a3+302660/2\
     1cdb2+4197721018/9261cb4-128/7cb2a-28823/12db5+99991009/63db3a-114271928/\
     441dba2+49/4b7+4b5a+86468/7b3a2-3457184/1029ba3-506216/63cdb+8393713780/9\
     261cb3-256/7cba-129625/126db4+432008/3087db2a+133/4b6+8b4a+24712/7b2a2-34\
     56512/1029a3-494080/63cd+2395735192/9261cb2-86405/63db3+2799920668/3087db\
     a-2b5+16/7b3a+49408/7ba2+518128cb-9766570/9261db2+9b4+32/7b2a-17280544/92\
     61db-36/7b3-17282560/9261d-44/7b2
  ↦ J[13]=b7a-296432/7da3+109/16b8+2b6a-2016b4a2+4610560/16807b2a3-14521796/3\
     087cdb2-22401156488/151263cb4+8000/343cb2a+1382867/1764db5-1600806884/308\
     7db3a+4148384/49dba2+77/4b7-194/49b5a-4032b3a2+18438656/16807ba3+8103808/\
     3087cdb-44793095440/151263cb3+13568/343cba+230189/686db4-6908840/151263db\
     2a+533/28b6-312/49b4a-1152b2a2+18435072/16807a3+7901200/3087cd-4261620128\
     /50421cb2-4864/343ca+1382435/3087db3-4980595264/16807dba+2078/49b5+176/34\
     3b3a-2304ba2-8294976/49cb+52056614/151263db2+2435/49b4-1472/343b2a+921811\
     84/151263db+1900/343b3+608/343ba+30725120/50421d+8044/343b2-2432/343b
  ↦ J[14]=b9+1037536/343da3+4b8+144b4a2-329408/16807b2a3+148256/441cdb2+15987\
     36592/151263cb4-24704/441db5+799730504/21609db3a-296064/49dba2+40/7b7+288\
     b3a2-1317376/16807ba3-4047040/21609cdb+3196814624/151263cb3+2048/2401cba-\
     24db4+164704/50421db2a+40/7b6-64/343b4a+576/7b2a2-3840/49a3-3950080/21609\
     cd+886720/147cb2+4096/2401ca-98816/3087db3+1036320/49dba+272/49b5-128/343\
     b3a+1152/7ba2+592000/49cb-3721504/151263db2+1152/343b4-256/2401b2a-658688\
     0/151263db+1024/343b3-512/2401ba-6400/147d+1920/2401b2+2048/2401b
  ↦ J[15]=dba3-7cb4-49/2db3a+4dba2-14cb3-4cb2-14dba-8cb
  ↦ J[16]=db3a2+1323da3+63b4a2-60/7b2a3+147cdb2+194441/42cb4-49/2db5+64843/4d\
     b3a-2646dba2+126b3a2-240/7ba3-82cdb+194401/21cb3-21/2db4+10/7db2a+36b2a2-\
     240/7a3-80cd+55486/21cb2-14db3+64847/7dba+72ba2+5292cb-226/21db2-400/21db\
     -400/21d
  ↦ J[17]=a5+4a4-7/6cb3+5/9da2-7/18b5-2/3cb+20/9da-2/9b3
  ↦ J[18]=da4-2/49b2a3-2/441cb4-1/63db3a-4/49ba3+10/63cdb-4/441cb3+32/7cba+1/\
     147db2a-b4a+20/63cd+64/7ca-2b3a-10/441db2+4b4-4/7b2a-20/441db+8b3-8/7ba+1\
     6/7b2+32/7b
```

See also: Section D.2.6.1 [modular], page 883.

D.15.11 multigrading_lib

Todos/Issues:

See `http://code.google.com/p/convex-singular/wiki/Multigrading`

Library: multigrading.lib

Purpose: Multigraded Rings

Authors: Benjamin Bechtold, benjamin.bechtold@googlemail.com
Rene Birkner, rbirkner@math.fu-berlin.de
Lars Kastner, lkastner@math.fu-berlin.de
Simon Keicher, keicher@mail.mathematik.uni-tuebingen.de
Oleksandr Motsak, U@D, where U={motsak}, D={mathematik.uni-kl.de}
Anna-Lena Winz, anna-lena.winz@math.fu-berlin.de

Overview: This library allows one to virtually add multigradings to Singular: grade multivariate polynomial rings with arbitrary (fin. gen. Abelian) groups. For more see http://code.google.com/p/convex-singular/wiki/Multigrading For theoretical references see:
E. Miller, B. Sturmfels: 'Combinatorial Commutative Algebra' and
M. Kreuzer, L. Robbiano: 'Computational Commutative Algebra'.

Note: 'multiDegBasis' relies on 4ti2 for computing Hilbert Bases. All groups are finitely generated Abelian

Procedures:

D.15.11.1 setBaseMultigrading

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: setBaseMultigrading(M[, G]); M is an integer matrix, G is a group (or lattice)

Purpose: attaches weights of variables and grading group to the basering.

Note: M encodes the weights of variables column-wise.

Return: nothing

Example:

```
LIB "multigrading.lib";
ring R = 0, (x, y, z), dp;
// Weights of variables
intmat M[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
// GradingGroup:
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z^3/L
// Weights are accessible via "getVariableWeights()":
getVariableWeights();
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
// Test all possible usages:
(getVariableWeights() == M) && (getVariableWeights(R) == M) && (getVariableWeights(ba
↦ 1
// Grading group is accessible via "getLattice()":
getLattice();
↦ 1,1,
↦ 1,3,
↦ 1,5
// Test all possible usages:
(getLattice() == L) && (getLattice(R) == L) && (getLattice(basering) == L);
↦ 1
// And its hermite NF via getLattice("hermite"):
getLattice("hermite");
↦ 1,0,
↦ 1,2,
↦ 1,4
// Test all possible usages:
intmat H = hermiteNormalForm(L);
(getLattice("hermite") == H) && (getLattice(R, "hermite") == H) && (getLattice(baseri
↦ 1
kill L, M;
// ----------- isomorphic multigrading -------- //
// Weights of variables
intmat M[2][3] =
1, -2, 1,
1,  1, 0;
// Torsion:
intmat L[2][1] =
0,
2;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z + (Z/2Z)
// Weights are accessible via "getVariableWeights()":
getVariableWeights() == M;
↦ 1
// Torsion is accessible via "getLattice()":
getLattice() == L;
↦ 1
kill L, M;
// ----------- extreme case ------------ //
// Weights of variables
intmat M[1][3] =
1,  -1, 10;
// Torsion:
intmat L[1][1] =
0;
// attaches M & L to R (==basering):
setBaseMultigrading(M); // Grading: Z^3
// Weights are accessible via "getVariableWeights()":
getVariableWeights() == M;
↦ 1
// Torsion is accessible via "getLattice()":
getLattice() == L;
↦ 1
```

D.15.11.2 getVariableWeights

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: getVariableWeights([R])

Purpose: get associated multigrading matrix for the basering [or R]

Return: intmat, matrix of multidegrees of variables

Example:

```
LIB "multigrading.lib";
ring R = 0, (x, y, z), dp;
// Weights of variables
intmat M[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
// Grading group:
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z^3/L
// Weights are accessible via "getVariableWeights()":
getVariableWeights() == M;
↦ 1
kill L, M;
// ----------- isomorphic multigrading -------- //
// Weights of variables
intmat M[2][3] =
1, -2, 1,
1,  1, 0;
// Grading group:
intmat L[2][1] =
0,
2;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z + (Z/2Z)
// Weights are accessible via "getVariableWeights()":
getVariableWeights() == M;
↦ 1
kill L, M;
// ----------- extreme case ------------ //
// Weights of variables
intmat M[1][3] =
1,  -1, 10;
// Grading group:
intmat L[1][1] =
0;
// attaches M & L to R (==basering):
setBaseMultigrading(M); // Grading: Z^3
// Weights are accessible via "getVariableWeights()":
getVariableWeights() == M;
```

```
↦ 1
```

D.15.11.3 getGradingGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: getGradingGroup([R])

Purpose: get associated grading group

Return: group, the grading group

Example:

```
LIB "multigrading.lib";
ring R = 0, (x, y, z), dp;
// Weights of variables
intmat M[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
// Torsion:
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z^3/L
def G = getGradingGroup();
printGroup( G );
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      1     1
↦      1     3
↦      1     5
G[1] == M; G[2] == L;
↦ 1
↦ 1
kill L, M, G;
// ----------- isomorphic multigrading -------- //
// Weights of variables
intmat M[2][3] =
1, -2, 1,
1,  1, 0;
// Torsion:
intmat L[2][1] =
0,
2;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z + (Z/2Z)
def G = getGradingGroup();
printGroup( G );
↦ Generators:
```

```
↦        1    -2     1
↦        1     1     0
↦ Relations:
↦        0
↦        2
G[1] == M; G[2] == L;
↦ 1
↦ 1
kill L, M, G;
// ----------- extreme case ------------ //
// Weights of variables
intmat M[1][3] =
1,  -1, 10;
// Torsion:
intmat L[1][1] =
0;
// attaches M & L to R (==basering):
setBaseMultigrading(M); // Grading: Z^3
def G = getGradingGroup();
printGroup( G );
↦ Generators:
↦        1    -1    10
↦ Relations:
↦        0
G[1] == M; G[2] == L;
↦ 1
↦ 1
kill L, M, G;
```

D.15.11.4 getLattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: getLattice([R[,opt]])

Purpose: get associated grading group matrix, i.e. generators (cols) of the grading group

Return: intmat, the grading group matrix, or
its hermite normal form if an optional argument ("hermiteNormalForm") is given or
smith normal form if an optional argument ("smith") is given

Example:

```
LIB "multigrading.lib";
ring R = 0, (x, y, z), dp;
// Weights of variables
intmat M[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
// Torsion:
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
// attaches M & L to R (==basering):
```

```
setBaseMultigrading(M, L); // Grading: Z^3/L
// Torsion is accessible via "getLattice()":
getLattice() == L;
↦ 1
// its hermite NF:
print(getLattice("hermite"));
↦      1     0
↦      1     2
↦      1     4
kill L, M;
// ----------- isomorphic multigrading -------- //
// Weights of variables
intmat M[2][3] =
1, -2, 1,
1,  1, 0;
// Torsion:
intmat L[2][1] =
0,
2;
// attaches M & L to R (==basering):
setBaseMultigrading(M, L); // Grading: Z + (Z/2Z)
// Torsion is accessible via "getLattice()":
getLattice() == L;
↦ 1
// its hermite NF:
print(getLattice("hermite"));
↦      0
↦      2
kill L, M;
// ----------- extreme case ------------ //
// Weights of variables
intmat M[1][3] =
1,  -1, 10;
// Torsion:
intmat L[1][1] =
0;
// attaches M & L to R (==basering):
setBaseMultigrading(M); // Grading: Z^3
// Torsion is accessible via "getLattice()":
getLattice() == L;
↦ 1
// its hermite NF:
print(getLattice("hermite"));
↦      0
```

D.15.11.5 createGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: createGroup(S, L); S, L are integer matrices

Purpose: create the group of the form (S+L)/L, i.e.
S specifies generators, L specifies relations.

Return: group

Example:

```
LIB "multigrading.lib";
intmat S[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
def G = createGroup(S, L); // (S+L)/L
printGroup(G);
↦ Generators:
↦       1     0     0
↦       0     1     0
↦       0     0     1
↦ Relations:
↦       1     1
↦       1     3
↦       1     5
kill S, L, G;
/////////////////////////////////////////////////
intmat S[2][3] =
1, -2, 1,
1,  1, 0;
intmat L[2][1] =
0,
2;
def G = createGroup(S, L); // (S+L)/L
printGroup(G);
↦ Generators:
↦       1    -2     1
↦       1     1     0
↦ Relations:
↦       0
↦       2
kill S, L, G;
// ----------- extreme case ------------ //
intmat S[1][3] =
1,  -1, 10;
// Torsion:
intmat L[1][1] =
0;
def G = createGroup(S, L); // (S+L)/L
printGroup(G);
↦ Generators:
↦       1    -1    10
↦ Relations:
↦       0
```

D.15.11.6 createQuotientGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: createGroup(L); L is an integer matrix

Purpose: create the group of the form (I+L)/L,
where I is the square identity matrix of size nrows(L) x nrows(L)

Note: L specifies relations between free generators of Z^nrows(L)

Return: group

Example:

```
LIB "multigrading.lib";
intmat I[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
// The group Z^3 / L can be constructed as follows:
// shortcut:
def G = createQuotientGroup(L);
printGroup(G);
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      1     1
↦      1     3
↦      1     5
// the general way:
def GG = createGroup(I, L); // (I+L)/L
printGroup(GG);
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      1     1
↦      1     3
↦      1     5
```

D.15.11.7 createTorsionFreeGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: createTorsionFreeGroup(S); S is an integer matrix

Purpose: create the free subgroup generated by S within the
free Abelian group of rank nrows(S)

Return: group

Example:

```
LIB "multigrading.lib";
// ----------- extreme case ------------ //
intmat S[1][3] =
1,  -1, 10;
// Torsion:
intmat L[1][1] =
0;
// The free subgroup generated by elements of S within Z^1
// can be constructed as follows:
// shortcut:
def G = createTorsionFreeGroup(S);
printGroup(G);
↦ Generators:
↦      1    -1    10
↦ Relations:
↦      0
// the general way:
def GG = createGroup(S, L); // (S+L)/L
printGroup(GG);
↦ Generators:
↦      1    -1    10
↦ Relations:
↦      0
```

D.15.11.8 printGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: printGroup(G); G is a group

Purpose: prints the group G

Return: nothing

Example:

```
LIB "multigrading.lib";
intmat S[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
def G = createGroup(S, L); // (S+L)/L
printGroup(G);
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      1     1
↦      1     3
↦      1     5
```

D.15.11.9 isGroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isGroup(G); G a list

Purpose: checks whether G is a valid group

Note: G should be created by createGroup
(or createQuotientGroup, createTorsionFreeGroup)

Return: int, 1 if G is a valid group and 0 otherwise

Example:

```
LIB "multigrading.lib";
intmat S[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L[3][2] =
1, 1,
1, 3,
1, 5;
def G = createGroup(S, L); // (S+L)/L
isGroup(G);
↦ 1
printGroup(G);
↦ Generators:
↦    1   0   0
↦    0   1   0
↦    0   0   1
↦ Relations:
↦    1   1
↦    1   3
↦    1   5
```

D.15.11.10 isGroupHomomorphism

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isGoupHomomorphism(L1,L2,A); L1 and L2 are groups, A is an integer matrix

Purpose: checks whether A defines a group homomorphism phi: L1 –> L2

Return: int, 1 if A defines the homomorphism and 0 otherwise

Example:

```
LIB "multigrading.lib";
intmat L1[4][1]=
0,
0,
0,
2;
intmat L2[3][2]=
0, 0,
2, 0,
0, 3;
```

```
intmat A[3][4] =
1, 2, 3, 0,
7, 0, 0, 0,
1, 2, 0, 3;
print( A );
↦      1     2     3     0
↦      7     0     0     0
↦      1     2     0     3
isGroupHomomorphism(L1, L2, A);
↦ 1
intmat B[3][4] =
1, 2, 3, 0,
7, 0, 0, 0,
1, 2, 0, 2;
print( B );
↦      1     2     3     0
↦      7     0     0     0
↦      1     2     0     2
isGroupHomomorphism(L1, L2, B); // Not a homomorphism!
↦ 0
```

D.15.11.11 isGradedRingHomomorphism

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isGradedRingHomomorphism(R, f, A); ring R, ideal f, group homomorphism A

Purpose: test a multigraded group ring homomorphism defined by a ring map from R to the current ring, given by generators images f and a group homomorphism A between grading groups

Return: int, 1 for TRUE, 0 otherwise

Example:

```
LIB "multigrading.lib";
ring r = 0, (x, y, z), dp;
intmat S1[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L1[3][1] =
0,
0,
0;
def G1 = createGroup(S1, L1); // (S1 + L1)/L1
printGroup(G1);
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      0
↦      0
↦      0
setBaseMultigrading(S1, L1); // to change...
```

```
ring R = 0, (a, b, c), dp;
intmat S2[2][3] =
1, 0,
0, 1;
intmat L2[2][1] =
0,
2;
def G2 = createGroup(S2, L2);
printGroup(G2);
↦ Generators:
↦      1     0     0
↦      1     0     0
↦ Relations:
↦      0
↦      2
setBaseMultigrading(S2, L2); // to change...
map F = r, a, b, c;
intmat A[nrows(L2)][nrows(L1)] =
1, 0, 0,
3, 2, -6;
// graded ring homomorphism is given by (compatible):
print(F);
↦ F[1]=a
↦ F[2]=b
↦ F[3]=c
print(A);
↦      1     0     0
↦      3     2    -6
isGradedRingHomomorphism(r, ideal(F), A);
↦ 1
def h = createGradedRingHomomorphism(r, ideal(F), A);
print(h);
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=a
↦    _[2]=b
↦    _[3]=c
↦ [3]:
↦    1,0,0,
↦    3,2,-6
// not a homo..
intmat B[nrows(L2)][nrows(L1)] =
1, 1, 1,
0, 0, 0;
print(B);
↦      1     1     1
↦      0     0     0
isGradedRingHomomorphism(r, ideal(F), B); // FALSE: there is no such homomorphism!
```

```
↦ 0
// Therefore: the following command should return an error
// createGradedRingHomomorphism(r, ideal(F), B);
```

D.15.11.12 createGradedRingHomomorphism

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: createGradedRingHomomorphism(R, f, A); ring R, ideal f, group homomorphism A

Purpose: create a multigraded group ring homomorphism defined by a ring map from R to the current ring, given by generators images f and a group homomorphism A between grading groups

Return: graded ring homorphism

Example:

```
LIB "multigrading.lib";
ring r = 0, (x, y, z), dp;
intmat S1[3][3] =
1, 0, 0,
0, 1, 0,
0, 0, 1;
intmat L1[3][1] =
0,
0,
0;
def G1 = createGroup(S1, L1); // (S1 + L1)/L1
printGroup(G1);
↦ Generators:
↦      1     0     0
↦      0     1     0
↦      0     0     1
↦ Relations:
↦      0
↦      0
↦      0
setBaseMultigrading(S1, L1); // to change...
ring R = 0, (a, b, c), dp;
intmat S2[2][3] =
1, 0,
0, 1;
intmat L2[2][1] =
0,
2;
def G2 = createGroup(S2, L2);
printGroup(G2);
↦ Generators:
↦      1     0     0
↦      1     0     0
↦ Relations:
↦      0
↦      2
setBaseMultigrading(S2, L2); // to change...
map F = r, a, b, c;
```

```
intmat A[nrows(L2)][nrows(L1)] =
1, 0, 0,
3, 2, -6;
// graded ring homomorphism is given by (compatible):
print(F);
↦ F[1]=a
↦ F[2]=b
↦ F[3]=c
print(A);
↦      1     0     0
↦      3     2    -6
isGradedRingHomomorphism(r, ideal(F), A);
↦ 1
def h = createGradedRingHomomorphism(r, ideal(F), A);
print(h);
↦ [1]:
↦    // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ [2]:
↦    _[1]=a
↦    _[2]=b
↦    _[3]=c
↦ [3]:
↦    1,0,0,
↦    3,2,-6
```

D.15.11.13 setModuleGrading

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: setModuleGrading(m, G), m module/vector, G intmat

Purpose: attaches the multiweights of free module generators to 'm'

Warning: The method does not verify whether the multigrading makes the module/vector homogeneous. One can do that using isHomogeneous(m).

Example:

```
LIB "multigrading.lib";
ring R = 0, (x,y), dp;
intmat M[2][2]=
1, 1,
0, 2;
intmat T[2][5]=
1,  2,  3,  4, 0,
0, 10, 20, 30, 1;
setBaseMultigrading(M, T);
ideal I = x, y, xy^5;
intmat V = multiDeg(I);
// V == M; modulo T
print(V);
↦      1     1    10
↦        0      2     10
module S = syz(I);
S = setModuleGrading(S, V);
getModuleGrading(S) == V;
↦ 1
print(S);
↦ -y,x4y5,
↦ x, 0,
↦ 0, -1
vector v = getGradedGenerator(S, 1);
getModuleGrading(v) == V;
↦ 1
print( multiDeg(v) );
↦ 2,
↦ 2
isHomogeneous(S);
↦ 1
print( multiDeg(S) );
↦        2     10
↦        2     10
```

D.15.11.14 getModuleGrading

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: getModuleGrading(m), 'm' module/vector

Return: integer matrix of the multiweights of free module generators attached to 'm'

Example:

```
LIB "multigrading.lib";
ring R = 0, (x,y), dp;
intmat M[2][2]=
1, 1,
0, 2;
intmat T[2][5]=
1,  2,  3,  4, 0,
0, 10, 20, 30, 1;
setBaseMultigrading(M, T);
ideal I = x, y, xy^5;
isHomogeneous(I);
↦ 1
intmat V = multiDeg(I); print(V);
↦        1      1     10
↦        0      2     10
module S = syz(I); print(S);
↦ -y,x4y5,
↦ x, 0,
↦ 0, -1
S = setModuleGrading(S, V);
getModuleGrading(S) == V;
↦ 1
vector v = getGradedGenerator(S, 1);
getModuleGrading(v) == V;
```

```
↦ 1
isHomogeneous(v);
↦ 1
print( multiDeg(v) );
↦ 2,
↦ 2
isHomogeneous(S);
↦ 1
print( multiDeg(S) );
↦      2    10
↦      2    10
```

D.15.11.15 isSublattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isSublattice(L, S); L, S are of tpye intmat

Purpose: checks whether the lattice created by L is a
sublattice of the lattice created by S.
The procedure checks whether each generator of L is
contained in S.

Return: integer, 0 if false, 1 if true

Example:

```
LIB "multigrading.lib";
//ring R = 0,(x,y),dp;
intmat S2[3][3]=
0, 2, 3,
0, 1, 1,
3, 0, 2;
intmat S1[3][2]=
0, 6,
0, 2,
3, 4;
isSublattice(S1,S2); // Yes!
↦ 1
intmat S3[3][1] =
0,
0,
1;
not(isSublattice(S3,S2)); // Yes!
↦ 1
```

D.15.11.16 imageLattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: imageLattice(Q,L); Q and L are of type intmat

Purpose: compute an integral basis for the image of the
lattice L under the homomorphism of lattices Q.

Return: intmat

Example:

```
LIB "multigrading.lib";
intmat Q[2][3] =
1,2,3,
3,2,1;
intmat L[3][2] =
1,4,
2,5,
3,6;
// should be a 2x2 matrix with columns
// [2,-14], [0,36]
imageLattice(Q,L);
↦ 2,0,
↦ 22,36
```

D.15.11.17 intRank

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: intRank(A); intmat A

Purpose: compute the rank of the integral matrix A
by computing a hermite normalform.

Returns: int

Example:

```
LIB "multigrading.lib";
intmat A[3][4] =
1,0,1,0,
1,2,0,0,
0,0,0,0;
int r = intRank(A);
print(A);
↦      1     0     1     0
↦      1     2     0     0
↦      0     0     0     0
print(r); // Should be 2
↦ 2
// another example
intmat B[2][2] =
1,2,
1,2;
int d = intRank(B);
print(B);
↦      1     2
↦      1     2
print(d); // Should be 1
↦ 1
kill A, B, r, d;
```

D.15.11.18 kernelLattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: kernelLattice(P); intmat P

Purpose: compute a integral basis for the kernel of the
homomorphism of lattices defined by the intmat P.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat LL[3][4] =
1,4,7,10,
2,5,8,11,
3,6,9,12;
// should be a 4x2 matrix whose columns
// generate the same lattice as [-1,2,-1,0],[2,-3,0,1]
intmat B = kernelLattice(LL);
print(B);
↦     -1     2
↦      2    -3
↦     -1     0
↦      0     1
// another example
intmat C[2][4] =
1,0,2,0,
0,1,2,0;
// should result in a matrix whose
// columns create the same  lattice as
// [-2,-2,1,0], [0,0,0,1]
intmat D = kernelLattice(C);
print(D);
↦     -2     0
↦     -2     0
↦      1     0
↦      0     1
kill B;
```

D.15.11.19 latticeBasis

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: latticeBasis(B); intmat B

Purpose: compute an integral basis for the lattice defined by the columns of B.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat L[3][3] =
1,4,8,
2,5,10,
3,6,12;
intmat B = latticeBasis(L);
print(B); // should result in a matrix whose columns generate the same lattice as  [1
↦      1     0
↦      2     3
↦      3     6
// another example
```

```
intmat C[2][4] =
1,1,2,0,
2,3,4,0;
// should result in a matrix whose
// columns create the same  lattice as
// [0,1],[1,0]
intmat D = latticeBasis(C);
print(D);
↦      1     0
↦      0     1
kill B,L;
```

D.15.11.20 preimageLattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: preimageLattice(P, B); intmat P, intmat B

Purpose: compute an integral basis for the preimage of B under the homomorphism of lattices defined by the intmat P.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat P[2][3] =
2,6,10,
4,8,12;
intmat B[2][1] =
1,
0;
// should be a (3x2)-matrix with columns e.g. [1,1,-1] and [0,3,-2] (the generated la
print(preimageLattice(P,B));
↦      1     0
↦      1     3
↦     -1    -2
// another example
intmat W[3][3] =
1,0,0,
0,1,1,
0,2,0;
intmat Z[3][2] =
1,0,
0,1,
0,0;
// should be a (3x2)-matrix with columns e.g. [1,0,0] and [0,0,-1] (the generated lat
print(preimageLattice(W,Z));
↦      1     0
↦      0     0
↦      0     1
```

D.15.11.21 projectLattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: projectLattice(B); intmat B

Purpose: A set of vectors in ZZ^m is given as the columns of B. Then this function provides a linear map ZZ^m –> ZZ^n having the primitive span of B its kernel.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat B[4][2] =
1,5,
2,6,
3,7,
4,8;
// should result in a (2x4)-matrix such that the corresponding lattice is created by
// [-1, 2], [-2, 3], [-1, 0] and [0, 1]
print(projectLattice(B));
↦     -1     2    -1     0
↦      2    -3     0     1
// another example
intmat BB[4][2] =
1,0,
0,1,
0,0,
0,0;
// should result in a (2x4)-matrix such that the corresponding lattice is created by
// [0,0],[0,0],[1,0],[0,1]
print(projectLattice(BB));
↦      0     0     1     0
↦      0     0     0     1
// another example
intmat BBB[3][4] =
1,0,1,2,
1,1,0,0,
3,0,0,3;
// should result in the (1x3)-matrix that consists of just zeros
print(projectLattice(BBB));
↦      0     0     0
```

D.15.11.22 intersectLattices

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: intersectLattices(A, B); intmat A, intmat B

Purpose: compute an integral basis for the intersection of the lattices A and B.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat A[3][2] =
1,4,
2,5,
3,6;
intmat B[3][2] =
6,9,
7,10,
8,11;
// should result in a (3x2)-matrix with columns
//  e.g. [0, 3, 6], [-3, 0, 3] (the lattice should be the same)
print(intersectLattices(A,B));
↦      3     0
↦      0     3
↦     -3     6
// another example
intmat C[2][3] =
1,0,0,
3,2,5;
intmat D[2][3] =
4,5,0,
0,5,0;
// should result in a (3x2)-matrix whose columns generate the
// same lattice as [1,5], [0, 20]
print(intersectLattices(C,D));
↦      1     0
↦      5    20
```

D.15.11.23 isIntegralSurjective

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: test whether the given linear map P of lattices is
surjective.

Returns: int, where 0 is false and 1 is true.

Example:

```
LIB "multigrading.lib";
intmat A[2][3] =
1,3,5,
2,4,6;
// should be 0
int b = isIntegralSurjective(A);
print(b);
↦ 0
// another example
intmat B[2][3] =
1,1,5,
2,3,6;
// should be 1
int c = isIntegralSurjective(B);
print(c);
↦ 1
kill A, b, B, c;
```

D.15.11.24 isPrimitiveSublattice

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: check whether the given set of integral vectors in ZZ^m, i.e. the columns of A, generate a primitive sublattice in ZZ^m (a direct summand of ZZ^m).

Returns: int, where 0 is false and 1 is true.

Example:

```
LIB "multigrading.lib";
intmat A[3][2] =
1,4,
2,5,
3,6;
// should be 0
int b = isPrimitiveSublattice(A);
print(b);
↦ 0
// another example
intmat B[2][2] =
1,0,
0,1;
// should be 1
int c = isPrimitiveSublattice(B);
print(c);
↦ 1
kill A, b, B, c;
```

D.15.11.25 intInverse

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: compute the integral inverse of the intmat A.
If det(A) is neither 1 nor -1 an error is returned.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat A[3][3] =
1,1,3,
3,2,0,
0,0,1;
intmat B = intInverse(A);
// should be the unit matrix
print(A * B);
↦      1     0     0
↦      0     1     0
↦      0     0     1
// another example
intmat C[2][2] =
2,1,
3,2;
intmat D  = intInverse(C);
// should be the unit matrix
print(C * D);
↦      1     0
↦      0     1
kill A, B, C, D;
```

D.15.11.26 integralSection

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: for a given linear surjective map P of lattices
this procedure returns an integral section of P.

Returns: intmat

Example:

```
LIB "multigrading.lib";
intmat P[2][4] =
1,3,4,6,
2,4,5,7;
// should be a matrix with two columns
// for example: [-2, 1, 0, 0], [3, -3, 0, 1]
intmat U = integralSection(P);
print(U);
↦     -1     1
↦     -2     1
↦      2    -1
↦      0     0
print(P * U);
↦      1     0
↦      0     1
kill U;
```

D.15.11.27 primitiveSpan

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: compute an integral basis for the minimal primitive
sublattice that contains the given vectors, i.e. the columns of V.

Returns: int, where 0 is false and 1 is true.

Example:

```
LIB "multigrading.lib";
intmat V[3][2] =
1,4,
2,5,
3,6;
// should return a (3x2)-matrix whose columns
// generate the same lattice as [1, 2, 3] and [0, 1, 2]
intmat R = primitiveSpan(V);
print(R);
↦      1     0
↦      2     1
↦      3     2
// another example
intmat W[2][2] =
1,0,
0,1;
// should return a (2x2)-matrix whose columns
// generate the same lattice as [1, 0] and [0, 1]
intmat S = primitiveSpan(W);
```

```
print(S);
↦      1     0
↦      0     1
kill V, R, S, W;
```

D.15.11.28 factorgroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: factorgroup(G,H); list G, list H

Purpose: returns a representation of the factor group G mod H using the first isomorphism thm

Returns: list

Example:

```
LIB "multigrading.lib";
intmat S1[2][2] =
1,0,
0,1;
intmat L1[2][1] =
2,
0;
intmat S2[2][1] =
1,
0;
intmat L2[2][1] =
2,
0;
list G = createGroup(S1,L1);
list H = createGroup(S2,L2);
list N = factorgroup(G,H);
print(N);
↦ [1]:
↦    1,0,
↦    0,1
↦ [2]:
↦    2,1,
↦    0,0
kill G,H,N,S1,L1,S2,L2;
```

D.15.11.29 productgroup

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: productgroup(G,H); list G, list H

Purpose: Returns a representation of the group G x H

Returns: list

Example:

```
LIB "multigrading.lib";
intmat S1[2][2] =
1,0,
0,1;
intmat L1[2][1] =
2,
0;
intmat S2[2][2] =
1,0,
0,2;
intmat L2[2][1] =
0,
3;
list G = createGroup(S1,L1);
list H = createGroup(S2,L2);
list N = productgroup(G,H);
print(N);
↦ [1]:
↦    1,0,0,0,
↦    0,1,0,0,
↦    0,0,1,0,
↦    0,0,0,2
↦ [2]:
↦    2,0,
↦    0,0,
↦    0,0,
↦    0,3
kill G,H,N,S1,L1,S2,L2;
```

D.15.11.30 multiDeg

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDeg(A); polynomial/vector/ideal/module A

Purpose: compute multidegree

Example:

```
LIB "multigrading.lib";
ring r = 0,(x, y), dp;
intmat A[2][2] = 1, 0, 0, 1;
print(A);
↦      1     0
↦      0     1
intmat Ta[2][1] = 0, 3;
print(Ta);
↦      0
↦      3
//   attrib(A, "gradingGroup", Ta); // to think about
//  "poly:";
setBaseMultigrading(A);
multiDeg( x*x, A );
↦ 2,0
multiDeg( y*y*y, A );
↦ 0,3
setBaseMultigrading(A, Ta);
multiDeg( x*x*y );
↦ 2,1
multiDeg( y*y*y*x );
```

```
↦ 1,3
multiDeg( x*y + x + 1 );
↦ 1,1
multiDegPartition(x*y + x + 1);
↦ _[1]=xy
↦ _[2]=x
↦ _[3]=1
print ( multiDeg(0) );
↦ 0,
↦ 0
poly zero = 0;
print ( multiDeg(zero) );
↦ 0,
↦ 0
//  "ideal:";
ideal I = y*x*x, x*y*y*y;
print( multiDeg(I) );
↦      2     1
↦      1     3
print ( multiDeg(ideal(0)) );
↦ 0,
↦ 0
print ( multiDeg(ideal(0,0,0)) );
↦      0     0     0
↦      0     0     0
//  "vectors:";
intmat B[2][2] = 0, 1, 1, 0;
print(B);
↦      0     1
↦      1     0
multiDeg( setModuleGrading(y*y*y*gen(2), B ));
↦ 1,3
multiDeg( setModuleGrading(x*x*gen(1), B ));
↦ 2,1
vector V = x*gen(1) + y*gen(2);
V = setModuleGrading(V, B);
multiDeg( V );
↦ 1,1
vector v1 = setModuleGrading([0, 0, 0], B);
print( multiDeg( v1 ) );
↦ 0,
↦ 0
vector v2 = setModuleGrading([0], B);
print( multiDeg( v2 ) );
↦ 0,
↦ 0
//  "module:";
module D = x*gen(1), y*gen(2);
D;
↦ D[1]=x*gen(1)
↦ D[2]=y*gen(2)
D = setModuleGrading(D, B);
print( multiDeg( D ) );
```

```
↦       1      1
↦       1      1
module DD = [0, 0],[0, 0, 0];
DD = setModuleGrading(DD, B);
print( multiDeg( DD ) );
↦       0      0
↦       0      0
module DDD = [0, 0];
DDD = setModuleGrading(DDD, B);
print( multiDeg( DDD ) );
↦ 0,
↦ 0
```

D.15.11.31 multiDegBasis

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegBasis(d), multidegree: intvec d

Assume: current ring is multigraded, monomial ordering is global

Purpose: compute all monomials of multidegree d

Example:

```
LIB "multigrading.lib";
ring R = 0, (x, y), dp;
intmat g1[2][2]=1,0,0,1;
intmat l[2][1]=2,0;
intmat g2[2][2]=1,1,1,1;
intvec v1=4,0;
intvec v2=4,4;
intmat g3[1][2]=1,1;
setBaseMultigrading(g3);
intvec v3=4:1;
v3;
↦ 4
multiDegBasis(v3);
↦ _[1]=x4
↦ _[2]=y4
↦ _[3]=xy3
↦ _[4]=x2y2
↦ _[5]=x3y
setBaseMultigrading(g1,l);
multiDegBasis(v1);
↦ _[1]=1
setBaseMultigrading(g2);
multiDegBasis(v2);
↦ _[1]=y4
↦ _[2]=x4
↦ _[3]=x3y
↦ _[4]=x2y2
↦ _[5]=xy3
intmat M[2][2] = 1, -1, -1, 1;
intvec d = -2, 2;
setBaseMultigrading(M);
```

```
multiDegBasis(d);
↦ _[1]=y2
attrib(_, "ZeroPart");
↦ _[1]=xy
kill R, M, d;
ring R = 0, (x, y, z), dp;
intmat M[2][3] = 1, -2, 1,     1, 1, 0;
intmat L[2][1] = 0, 2;
intvec d = 4, 1;
setBaseMultigrading(M, L);
multiDegBasis(d);
↦ _[1]=x3z
↦ _[2]=x6y
↦ _[3]=yz6
↦ _[4]=xz3
attrib(_, "ZeroPart");
↦ _[1]=xyz
↦ _[2]=x4y2
↦ _[3]=y2z4
kill R, M, d;
ring R = 0, (x, y, z), dp;
qring Q = std(ideal( y^6+ x*y^3*z-x^2*z^2 ));
intmat M[2][3] = 1, 1, 2,     2, 1, 1;
//  intmat T[2][1] = 0, 2;
setBaseMultigrading(M); // BUG????
intvec d = 6, 6;
multiDegBasis(d);
↦ _[1]=x2z2
↦ _[2]=xy3z
attrib(_, "ZeroPart");
↦ _[1]=0
kill R, Q, M, d;
ring R = 0, (x, y, z), dp;
qring Q = std(ideal( x*z^3 - y *z^6, x*y*z  - x^4*y^2 ));
intmat M[2][3] = 1, -2, 1,     1, 1, 0;
intmat T[2][1] = 0, 2;
intvec d = 4, 1;
setBaseMultigrading(M, T); // BUG????
multiDegBasis(d);
↦ _[1]=x3z
↦ _[2]=x6y
↦ _[3]=xz3
attrib(_, "ZeroPart");
↦ _[1]=xyz
↦ _[2]=y2z4
```

D.15.11.32 multiDegPartition

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegPartition(def p), p polynomial/vector

Returns: an ideal/module consisting of multigraded-homogeneous parts of p

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
intmat g[2][3]=
1,0,1,
0,1,1;
intmat t[2][1]=
-2,
1;
setBaseMultigrading(g,t);
poly f = x10yz+x8y2z-x4z2+y5+x2y2-z2+x17z3-y6;
multiDegPartition(f);
↦ _[1]=x17z3
↦ _[2]=x10yz+x8y2z
↦ _[3]=-y6
↦ _[4]=-x4z2+y5
↦ _[5]=x2y2-z2
vector v = xy*gen(1)-x3y2*gen(2)+x4y*gen(3);
intmat B[2][3]=1,-1,-2,0,0,1;
v = setModuleGrading(v,B);
getModuleGrading(v);
↦ 1,-1,-2,
↦ 0,0,1
multiDegPartition(v, B);
↦ _[1]=x4y*gen(3)-x3y2*gen(2)
↦ _[2]=xy*gen(1)
```

D.15.11.33 isTorsionFree

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isTorsionFree()

Purpose: Determines whether the multigrading attached to the current ring is free.

Return: boolean, the result of the test

Example:

```
LIB "multigrading.lib";
ring R = 0,(x,y),dp;
intmat M[2][2]=
1,0,
0,1;
intmat T[2][5]=
1, 2, 3, 4, 0,
0,10,20,30, 1;
setBaseMultigrading(M,T);
// Is the resulting group  free?
isTorsionFree();
↦ 1
kill R, M, T;
///////////////////////////////////////////
ring R=0,(x,y,z),dp;
intmat A[3][3] =
1,0,0,
0,1,0,
0,0,1;
intmat B[3][4]=
3,3,3,3,
2,1,3,0,
1,2,0,3;
setBaseMultigrading(A,B);
// Is the resulting group  free?
isTorsionFree();
↦ 0
kill R, A, B;
```

D.15.11.34 isPositive

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isPositive()

Purpose: Computes whether the multigrading of the ring is positive. For computation theorem 8.6 of the Miller/Sturmfels book is used.

Returns: true if the multigrading is positive

Example:

```
LIB "multigrading.lib";
printlevel = 3;
ring r = 0,(x,y),dp;
intmat A[1][2]=-1,1;
setBaseMultigrading(A);
isPositive();
↦ 0
intmat B[1][2]=1,1;
setBaseMultigrading(B);
isPositive(B);
↦ 1
```

D.15.11.35 isZeroElement

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isZeroElement(d, [T]); intvec d, group T

Purpose: For a integer vector 'd' representing the multidegree of some polynomial or vector this method computes if the multidegree is contained in the grading group group (either set globally or given as an optional argument), i.e. if it is zero in the multigrading.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
intmat g[2][3]=
1,0,1,
0,1,1;
intmat t[2][1]=
-2,
1;
intmat tt[2][1]=
1,
-1;
setBaseMultigrading(g,t);
poly a = x10yz;
poly b = x8y2z;
poly c = x4z2;
poly d = y5;
poly e = x2y2;
poly f = z2;
intvec v1 = multiDeg(a) - multiDeg(b);
v1;
↦ 2,-1
isZeroElement(v1);
↦ 1
isZeroElement(v1, tt);
↦ 0
intvec v2 = multiDeg(a) - multiDeg(c);
v2;
↦ 5,0
isZeroElement(v2);
↦ 0
isZeroElement(v2, tt);
↦ 0
intvec v3 = multiDeg(e) - multiDeg(f);
v3;
↦ 0,0
isZeroElement(v3);
↦ 1
isZeroElement(v3, tt);
↦ 1
intvec v4 = multiDeg(c) - multiDeg(d);
v4;
↦ 6,-3
isZeroElement(v4);
↦ 1
isZeroElement(v4, tt);
↦ 0
```

D.15.11.36 areZeroElements

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: areZeroElements(D, [T]); intmat D, group T

Purpose: For a integer matrix D, considered column-wise as a set of integer vecors representing the multidegree of some polynomial or vector this method checks whether all these multidegrees are contained in the grading group
group (either set globally or given as an optional argument), i.e. if they all are zero in the multigrading.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
intmat S[2][3]=
1,0,1,
0,1,1;
intmat L[2][1]=
2,
2;
setBaseMultigrading(S,L);
poly a = 1;
poly b = xyz;
ideal I = a, b;
print(multiDeg(I));
↦      0     2
↦      0     2
intmat m[5][2]=multiDeg(a),multiDeg(b); m=transpose(m);
print(multiDeg(a));
↦ 0,
↦ 0
print(multiDeg(b));
↦ 2,
↦ 2
print(m);
↦      0     2     0     0     0
↦      0     2     0     0     0
areZeroElements(m);
↦ 1
intmat LL[2][1]=
1,
-1;
areZeroElements(m,LL);
↦ 0
```

D.15.11.37 isHomogeneous

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: isHomogeneous(a[, f]); a polynomial/vector/ideal/module

Return: boolean, TRUE if a is (multi)homogeneous, and FALSE otherwise

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
//Grading and Torsion matrices:
intmat M[3][3] =
1,0,0,
0,1,0,
0,0,1;
intmat T[3][1] =
1,2,3;
setBaseMultigrading(M,T);
attrib(r);
↦ attr:cf_class, type int
↦ attr:global, type int
↦ attr:maxExp, type int
↦ attr:ring_cf, type int
↦ attr:isLetterplaceRing, type int
↦ attr:gradingGroup, type list
↦ attr:mgrad, type intmat
poly f = x-yz;
multiDegPartition(f);
↦ _[1]=-yz
↦ _[2]=x
print(multiDeg(_));
↦      0     1
↦      1     0
↦      1     0
isHomogeneous(f);   // f: is not homogeneous
↦ 0
poly g = 1-xy2z3;
isHomogeneous(g); // g: is homogeneous
↦ 1
multiDegPartition(g);
↦ _[1]=-xy2z3+1
kill T;
/////////////////////////////////////////////////////////
// new Torsion matrix:
intmat T[3][4] =
3,3,3,3,
2,1,3,0,
1,2,0,3;
setBaseMultigrading(M,T);
f;
↦ -yz+x
isHomogeneous(f);
↦ 0
multiDegPartition(f);
↦ _[1]=-yz
↦ _[2]=x
// ---------------------
g;
↦ -xy2z3+1
isHomogeneous(g);
↦ 0
multiDegPartition(g);
↦ _[1]=-xy2z3
↦ _[2]=1
kill r, T, M;
ring R = 0, (x,y,z), dp;
intmat A[2][3] =
0,0,1,
3,2,1;
intmat T[2][1] =
-1,
4;
setBaseMultigrading(A, T);
isHomogeneous(ideal(x2 - y3 -xy +z, x*y-z, x^3 - y^2*z + x^2 -y^3)); // 1
↦ 1
isHomogeneous(ideal(x2 - y3 -xy +z, x*y-z, x^3 - y^2*z + x^2 -y^3), "checkGens");
↦ 0
```

```
isHomogeneous(ideal(x+y, x2 - y2)); // 0
↦ 0
// Degree partition:
multiDegPartition(x2 - y3 -xy +z);
↦ _[1]=-y3+x2
↦ _[2]=-xy+z
multiDegPartition(x3 -y2z + x2 -y3 + z + 1);
↦ _[1]=x3-y2z
↦ _[2]=-y3+x2
↦ _[3]=z
↦ _[4]=1
module N = gen(1) + (x+y) * gen(2), z*gen(3);
intmat V[2][3] = 0; // 1, 2, 3,  4, 5, 6; //  column-wise weights of components!!??
vector v1, v2;
v1 = setModuleGrading(N[1], V); v1;
↦ x*gen(2)+y*gen(2)+gen(1)
multiDegPartition(v1);
↦ _[1]=x*gen(2)
↦ _[2]=y*gen(2)
↦ _[3]=gen(1)
print( multiDeg(_) );
↦      0     0     0
↦      3     2     0
v2 = setModuleGrading(N[2], V); v2;
↦ z*gen(3)
multiDegPartition(v2);
↦ _[1]=z*gen(3)
print( multiDeg(_) );
↦      1
↦      1
N = setModuleGrading(N, V);
isHomogeneous(N);
↦ 0
print( multiDeg(N) );
↦      0     1
↦      3     1
///////////////////////////////////////
V =
1, 2, 3,
4, 5, 6;
v1 = setModuleGrading(N[1], V); v1;
↦ x*gen(2)+y*gen(2)+gen(1)
multiDegPartition(v1);
↦ _[1]=x*gen(2)
↦ _[2]=y*gen(2)
↦ _[3]=gen(1)
print( multiDeg(_) );
↦      2     2     1
↦      8     7     4
v2 = setModuleGrading(N[2], V); v2;
↦ z*gen(3)
multiDegPartition(v2);
↦ _[1]=z*gen(3)
```

```
print( multiDeg(_) );
↦      4
↦      7
N = setModuleGrading(N, V);
isHomogeneous(N);
↦ 0
print( multiDeg(N) );
↦      2     4
↦      8     7
///////////////////////////////////////
V =
0, 0, 0,
4, 1, 0;
N = gen(1) + x * gen(2), z*gen(3);
N = setModuleGrading(N, V); print(N);
↦ 1,0,
↦ x,0,
↦ 0,z
isHomogeneous(N);
↦ 1
print( multiDeg(N) );
↦      0     1
↦      4     1
v1 = getGradedGenerator(N,1); print(v1);
↦ [1,x]
multiDegPartition(v1);
↦ _[1]=x*gen(2)+gen(1)
print( multiDeg(_) );
↦      0
↦      4
N = setModuleGrading(N, V); print(N);
↦ 1,0,
↦ x,0,
↦ 0,z
isHomogeneous(N);
↦ 1
print( multiDeg(N) );
↦      0     1
↦      4     1
```

D.15.11.38 equalMultiDeg

Procedure from library multigrading.lib (see Section D.15.11 [multigrading_lib], page 2458).

Usage: equalMultiDeg(exp1, exp2[, V]); intvec exp1, exp2, intmat V

Purpose: Tests if the exponent vectors of two monomials (given by exp1 and exp2) represent the same multidegree.

Note: the integer matrix V encodes multidegrees of module components, if module component is present in exp1 and exp2

Example:

```
LIB "multigrading.lib";
printlevel=3;
```

```
ring r = 0,(x,y,z),dp;
intmat g[2][3]=
1,0,1,
0,1,1;
intmat t[2][1]=
-2,
1;
setBaseMultigrading(g,t);
poly a = x10yz;
poly b = x8y2z;
poly c = x4z2;
poly d = y5;
poly e = x2y2;
poly f = z2;
equalMultiDeg(leadexp(a), leadexp(b));
↦ 1
equalMultiDeg(leadexp(a), leadexp(c));
↦ 0
equalMultiDeg(leadexp(a), leadexp(d));
↦ 0
equalMultiDeg(leadexp(a), leadexp(e));
↦ 0
equalMultiDeg(leadexp(a), leadexp(f));
↦ 0
equalMultiDeg(leadexp(b), leadexp(c));
↦ 0
equalMultiDeg(leadexp(b), leadexp(d));
↦ 0
equalMultiDeg(leadexp(b), leadexp(e));
↦ 0
equalMultiDeg(leadexp(b), leadexp(f));
↦ 0
equalMultiDeg(leadexp(c), leadexp(d));
↦ 1
equalMultiDeg(leadexp(c), leadexp(e));
↦ 0
equalMultiDeg(leadexp(c), leadexp(f));
↦ 0
equalMultiDeg(leadexp(d), leadexp(e));
↦ 0
equalMultiDeg(leadexp(d), leadexp(f));
↦ 0
equalMultiDeg(leadexp(e), leadexp(f));
↦ 1
```

D.15.11.39 multiDegGroebner

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegGroebner(I); I is a poly/vector/ideal/module

Purpose: computes the multigraded standard/groebner basis of I

Note: I must be multigraded homogeneous

Returns: ideal/module, the computed basis

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z,w),dp;
intmat MM[2][4]=
1,1,1,1,
0,1,3,4;
setBaseMultigrading(MM);
module M = ideal(  xw-yz, x2z-y3, xz2-y2w, yw2-z3);
intmat v[2][nrows(M)]=
1,
0;
M = setModuleGrading(M, v);
/////////////////////////////////////////////////////////////////////////////
// GB:
M = multiDegGroebner(M); M;
↦ M[1]=yz*gen(1)-xw*gen(1)
↦ M[2]=z3*gen(1)-yw2*gen(1)
↦ M[3]=xz2*gen(1)-y2w*gen(1)
↦ M[4]=y3*gen(1)-x2z*gen(1)
"Module Units Multigrading: "; print( getModuleGrading(M) );
↦ Module Units Multigrading:
↦      1
↦      0
"Multidegrees: "; print(multiDeg(M));
↦ Multidegrees:
↦      3     4     4     4
↦      4     9     6     3
isHomogeneous(M);
↦ 1
/////////////////////////////////////////////////////////////////////////////
// Let’s compute Syzygy!
def S = multiDegSyzygy(M); S;
↦ S[1]=yw*gen(1)-x*gen(2)+z*gen(3)
↦ S[2]=z2*gen(1)-y*gen(2)+w*gen(3)
↦ S[3]=xz*gen(1)-y*gen(3)-w*gen(4)
↦ S[4]=y2*gen(1)-x*gen(3)-z*gen(4)
"Module Units Multigrading: "; print( getModuleGrading(S) );
↦ Module Units Multigrading:
↦      3     4     4     4
↦      4     9     6     3
"Multidegrees: "; print(multiDeg(S));
↦ Multidegrees:
↦      5     5     5     5
↦      9    10     7     6
isHomogeneous(S);
↦ 1
/////////////////////////////////////////////////////////////////////////////
// GB:
S = multiDegGroebner(S); S;
↦ S[1]=yw*gen(1)-x*gen(2)+z*gen(3)
↦ S[2]=z2*gen(1)-y*gen(2)+w*gen(3)
↦ S[3]=xz*gen(1)-y*gen(3)-w*gen(4)
↦ S[4]=y2*gen(1)-x*gen(3)-z*gen(4)
↦ S[5]=xy*gen(2)-yz*gen(3)-xw*gen(3)-zw*gen(4)
↦ S[6]=xz2*gen(2)-z3*gen(3)-y2w*gen(2)+yw2*gen(3)
↦ S[7]=x2z*gen(2)-xz2*gen(3)-y2w*gen(3)-yw2*gen(4)
↦ S[8]=y3*gen(2)-xz2*gen(3)-z3*gen(4)-y2w*gen(3)
↦ S[9]=y3*gen(3)-x2z*gen(3)-xz2*gen(4)+y2w*gen(4)
"Module Units Multigrading: "; print( getModuleGrading(S) );
↦ Module Units Multigrading:
↦      3    4    4    4
↦      4    9    6    3
"Multidegrees: "; print(multiDeg(S));
↦ Multidegrees:
↦      5    5    5    5    6    7    7    7    7
↦      9   10    7    6   10   15   12   12    9
isHomogeneous(S);
↦ 1
```

D.15.11.40 multiDegSyzygy

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegSyzygy(I); I is a ideal or a module

Purpose: computes the multigraded syzygy module of I

Returns: module, the syzygy module of I

Note: generators of I must be multigraded homogeneous

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z,w),dp;
intmat MM[2][4]=
1,1,1,1,
0,1,3,4;
setBaseMultigrading(MM);
module M = ideal(  xw-yz, x2z-y3, xz2-y2w, yw2-z3);
intmat v[2][nrows(M)]=
1,
0;
M = setModuleGrading(M, v);
isHomogeneous(M);
↦ 1
"Multidegrees: "; print(multiDeg(M));
↦ Multidegrees:
↦      3    4    4    4
↦      4    3    6    9
// Let's compute syzygies!
def S = multiDegSyzygy(M); S;
↦ S[1]=yw*gen(1)-x*gen(4)-z*gen(3)
↦ S[2]=z2*gen(1)-y*gen(4)-w*gen(3)
↦ S[3]=xz*gen(1)+y*gen(3)-w*gen(2)
↦ S[4]=y2*gen(1)+x*gen(3)-z*gen(2)
"Module Units Multigrading: "; print( getModuleGrading(S) );
```

```
↦ Module Units Multigrading:
↦       3     4     4     4
↦       4     3     6     9
"Multidegrees: "; print(multiDeg(S));
↦ Multidegrees:
↦       5     5     5     5
↦       9    10     7     6
isHomogeneous(S);
↦ 1
```

D.15.11.41 multiDegModulo

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegModulo(I); I, J are ideals or modules

Purpose: computes the multigraded 'modulo' module of I and J

Returns: module, see 'modulo' command

Note: I and J should have the same multigrading, and their generators must be multigraded homogeneous

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
intmat MM[2][3]=
-1,1,1,
0,1,3;
setBaseMultigrading(MM);
ideal h1 = x, y, z;
ideal h2 = x;
"Multidegrees: "; print(multiDeg(h1));
↦ Multidegrees:
↦      -1     1     1
↦       0     1     3
// Let's compute modulo(h1, h2):
def K = multiDegModulo(h1, h2); K;
↦ K[1]=gen(1)
↦ K[2]=y*gen(3)-z*gen(2)
↦ K[3]=x*gen(2)
↦ K[4]=x*gen(3)
"Module Units Multigrading: "; print( getModuleGrading(K) );
↦ Module Units Multigrading:
↦      -1     1     1
↦       0     1     3
"Multidegrees: "; print(multiDeg(K));
↦ Multidegrees:
↦      -1     2     0     0
↦       0     4     1     3
isHomogeneous(K);
↦ 1
```

D.15.11.42 multiDegResolution

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegResolution(I,l,[f]); I is poly/vector/ideal/module; l,f are integers

Purpose: computes the multigraded resolution of I of the length l, or the whole resolution if l is zero. Returns minimal resolution if an optional argument 1 is supplied

Note: input must have multigraded-homogeneous generators.
The returned list is truncated beginning with the first zero differential.

Returns: list, the computed resolution

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z,w),dp;
intmat M[2][4]=
1,1,1,1,
0,1,3,4;
setBaseMultigrading(M);
module m= ideal(  xw-yz, x2z-y3, xz2-y2w, yw2-z3);
isHomogeneous(ideal(  xw-yz, x2z-y3, xz2-y2w, yw2-z3), "checkGens");
↦ 1
ideal A = xw-yz, x2z-y3, xz2-y2w, yw2-z3;
int j;
for(j=1; j<=ncols(A); j++)
{
multiDegPartition(A[j]);
}
↦ _[1]=-yz+xw
↦ _[1]=-y3+x2z
↦ _[1]=xz2-y2w
↦ _[1]=-z3+yw2
intmat v[2][1]=
1,
0;
m = setModuleGrading(m, v);
// Let's compute Syzygy!
def S = multiDegSyzygy(m); S;
↦ S[1]=yw*gen(1)-x*gen(4)-z*gen(3)
↦ S[2]=z2*gen(1)-y*gen(4)-w*gen(3)
↦ S[3]=xz*gen(1)+y*gen(3)-w*gen(2)
↦ S[4]=y2*gen(1)+x*gen(3)-z*gen(2)
"Module Units Multigrading: "; print( getModuleGrading(S) );
↦ Module Units Multigrading:
↦      3    4    4    4
↦      4    3    6    9
"Multidegrees: "; print(multiDeg(S));
↦ Multidegrees:
↦      5    5    5    5
↦      9   10    7    6
/////////////////////////////////////////////////////////////////////////////
S = multiDegGroebner(S); S;
↦ S[1]=yw*gen(1)-x*gen(4)-z*gen(3)
↦ S[2]=z2*gen(1)-y*gen(4)-w*gen(3)
↦ S[3]=xz*gen(1)+y*gen(3)-w*gen(2)
↦ S[4]=y2*gen(1)+x*gen(3)-z*gen(2)
↦ S[5]=xy*gen(4)+yz*gen(3)+xw*gen(3)-zw*gen(2)
↦ S[6]=xz2*gen(4)+z3*gen(3)-y2w*gen(4)-yw2*gen(3)
↦ S[7]=x2z*gen(4)+xz2*gen(3)+y2w*gen(3)-yw2*gen(2)
↦ S[8]=y3*gen(3)-x2z*gen(3)+xz2*gen(2)-y2w*gen(2)
↦ S[9]=y3*gen(4)+xz2*gen(3)-z3*gen(2)+y2w*gen(3)
"Module Units Multigrading: "; print( getModuleGrading(S) );
↦ Module Units Multigrading:
↦      3     4     4     4
↦      4     3     6     9
"Multidegrees: "; print(multiDeg(S));
↦ Multidegrees:
↦      5     5     5     5     6     7     7     7     7
↦      9    10     7     6    10    15    12     9    12
///////////////////////////////////////////////////////////////////////////////
list L = multiDegResolution(m, 0, 1);
for( j =1; j<=size(L); j++)
{
"---------------------------------- ", j, " -----------------------------";
L[j];
"Module Multigrading: "; print( getModuleGrading(L[j]) );
"Multigrading: "; print(multiDeg(L[j]));
}
↦ ----------------------------------  1  ----------------------------
↦ _[1]=yz*gen(1)-xw*gen(1)
↦ _[2]=z3*gen(1)-yw2*gen(1)
↦ _[3]=xz2*gen(1)-y2w*gen(1)
↦ _[4]=y3*gen(1)-x2z*gen(1)
↦ Module Multigrading:
↦      1
↦      0
↦ Multigrading:
↦      3     4     4     4
↦      4     9     6     3
↦ ----------------------------------  2  ----------------------------
↦ _[1]=yw*gen(1)-x*gen(2)+z*gen(3)
↦ _[2]=z2*gen(1)-y*gen(2)+w*gen(3)
↦ _[3]=xz*gen(1)-y*gen(3)-w*gen(4)
↦ _[4]=y2*gen(1)-x*gen(3)-z*gen(4)
↦ Module Multigrading:
↦      3     4     4     4
↦      4     9     6     3
↦ Multigrading:
↦      5     5     5     5
↦      9    10     7     6
↦ ----------------------------------  3  ----------------------------
↦ _[1]=x*gen(2)-y*gen(1)-z*gen(3)+w*gen(4)
↦ Module Multigrading:
↦      5     5     5     5
↦      9    10     7     6
↦ Multigrading:
↦      6
↦     10
///////////////////////////////////////////////////////////////////////////////
L = multiDegResolution(maxideal(1), 0, 1);
```

```
for( j =1; j<=size(L); j++)
{
"----------------------------------- ", j, " -----------------------------";
L[j];
"Module Multigrading: "; print( getModuleGrading(L[j]) );
"Multigrading: "; print(multiDeg(L[j]));
}
↦ -----------------------------------  1  -----------------------------
↦ _[1]=w
↦ _[2]=z
↦ _[3]=y
↦ _[4]=x
↦ Module Multigrading:
↦      0
↦      0
↦ Multigrading:
↦      1     1     1     1
↦      4     3     1     0
↦ -----------------------------------  2  -----------------------------
↦ _[1]=-z*gen(1)+w*gen(2)
↦ _[2]=-y*gen(1)+w*gen(3)
↦ _[3]=-y*gen(2)+z*gen(3)
↦ _[4]=-x*gen(1)+w*gen(4)
↦ _[5]=-x*gen(2)+z*gen(4)
↦ _[6]=-x*gen(3)+y*gen(4)
↦ Module Multigrading:
↦      1     1     1     1
↦      4     3     1     0
↦ Multigrading:
↦      2     2     2     2     2     2
↦      7     5     4     4     3     1
↦ -----------------------------------  3  -----------------------------
↦ _[1]=y*gen(1)-z*gen(2)+w*gen(3)
↦ _[2]=x*gen(1)-z*gen(4)+w*gen(5)
↦ _[3]=x*gen(2)-y*gen(4)+w*gen(6)
↦ _[4]=x*gen(3)-y*gen(5)+z*gen(6)
↦ Module Multigrading:
↦      2     2     2     2     2     2
↦      7     5     4     4     3     1
↦ Multigrading:
↦      3     3     3     3
↦      8     7     5     4
↦ -----------------------------------  4  -----------------------------
↦ _[1]=-x*gen(1)+y*gen(2)-z*gen(3)+w*gen(4)
↦ Module Multigrading:
↦      3     3     3     3
↦      8     7     5     4
↦ Multigrading:
↦      4
↦      8
kill v;
def h = hilbertSeries(m);
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
setring h;
numerator1;
↦ -t_(1)^6*t_(2)^10+t_(1)^5*t_(2)^10+t_(1)^5*t_(2)^9-t_(1)^4*t_(2)^9+t_(1)^\
   5*t_(2)^7+t_(1)^5*t_(2)^6-t_(1)^4*t_(2)^6-t_(1)^4*t_(2)^3-t_(1)^3*t_(2)^4\
   +t_(1)
factorize(numerator1);
↦ [1]:
↦    _[1]=-1
↦    _[2]=t_(1)
↦    _[3]=t_(1)*t_(2)-1
↦    _[4]=t_(1)*t_(2)^3-1
↦    _[5]=t_(1)^3*t_(2)^6-t_(1)^2*t_(2)^6-t_(1)^2*t_(2)^2-t_(1)*t_(2)^3-t_(\
   1)*t_(2)-1
↦ [2]:
↦    1,1,1,1,1
denominator1;
↦ t_(1)^4*t_(2)^8-t_(1)^3*t_(2)^8-t_(1)^3*t_(2)^7+t_(1)^2*t_(2)^7-t_(1)^3*t\
   _(2)^5-t_(1)^3*t_(2)^4+t_(1)^2*t_(2)^5+2*t_(1)^2*t_(2)^4+t_(1)^2*t_(2)^3-\
   t_(1)*t_(2)^4-t_(1)*t_(2)^3+t_(1)^2*t_(2)-t_(1)*t_(2)-t_(1)+1
factorize(denominator1);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)-1
↦    _[3]=t_(1)*t_(2)-1
↦    _[4]=t_(1)*t_(2)^3-1
↦    _[5]=t_(1)*t_(2)^4-1
↦ [2]:
↦    1,1,1,1,1
numerator2;
↦ -t_(1)^4*t_(2)^6+t_(1)^3*t_(2)^6+t_(1)^3*t_(2)^2+t_(1)^2*t_(2)^3+t_(1)^2*\
   t_(2)+t_(1)
factorize(numerator2);
↦ [1]:
↦    _[1]=-1
↦    _[2]=t_(1)
↦    _[3]=t_(1)^3*t_(2)^6-t_(1)^2*t_(2)^6-t_(1)^2*t_(2)^2-t_(1)*t_(2)^3-t_(\
   1)*t_(2)-1
↦ [2]:
↦    1,1,1
denominator2;
↦ t_(1)^2*t_(2)^4-t_(1)*t_(2)^4-t_(1)+1
factorize(denominator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)-1
↦    _[3]=t_(1)*t_(2)^4-1
↦ [2]:
↦    1,1,1
```

D.15.11.43 multiDegTensor

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: multiDegTensor(m, n), m,n modules or matrices.

Purpose: Computes the multigraded tensor product of to multigraded modules.

Return: A module.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x),dp;
intmat g[2][1]=1,1;
setBaseMultigrading(g);
matrix m[5][3]=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15;
matrix n[3][2]=x,x2,x3,x4,x5,x6;
module mm = m;
module nn = n;
intmat gm[2][5]=1,2,3,4,5,0,0,0,0,0;
intmat gn[2][3]=0,0,0,1,2,3;
mm = setModuleGrading(mm, gm);
nn = setModuleGrading(nn, gn);
module mmtnn = multiDegTensor(mm, nn);
print(mmtnn);
↦ x, x2,0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3, 0, 0,
↦ x3,x4,0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3, 0,
↦ x5,x6,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3,
↦ 0, 0, x, x2,0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6, 0, 0,
↦ 0, 0, x3,x4,0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6, 0,
↦ 0, 0, x5,x6,0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6,
↦ 0, 0, 0, 0, x, x2,0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9, 0, 0,
↦ 0, 0, 0, 0, x3,x4,0, 0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9, 0,
↦ 0, 0, 0, 0, x5,x6,0, 0, 0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9,
↦ 0, 0, 0, 0, 0, 0, x, x2,0, 0, 10,0, 0, 11,0, 0, 12,0, 0,
↦ 0, 0, 0, 0, 0, 0, x3,x4,0, 0, 0, 10,0, 0, 11,0, 0, 12,0,
↦ 0, 0, 0, 0, 0, 0, x5,x6,0, 0, 0, 0, 10,0, 0, 11,0, 0, 12,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x, x2,13,0, 0, 14,0, 0, 15,0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x3,x4,0, 13,0, 0, 14,0, 0, 15,0,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x5,x6,0, 0, 13,0, 0, 14,0, 0, 15
getModuleGrading(mmtnn);
↦ 1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,
↦ 1,2,3,1,2,3,1,2,3,1,2,3,1,2,3
LIB "homolog.lib";
module tt = tensorMod(mm,nn);
print(tt);
↦ x, x2,0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3, 0, 0,
↦ x3,x4,0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3, 0,
↦ x5,x6,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3,
↦ 0, 0, x, x2,0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6, 0, 0,
↦ 0, 0, x3,x4,0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6, 0,
↦ 0, 0, x5,x6,0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 5, 0, 0, 6,
↦ 0, 0, 0, 0, x, x2,0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9, 0, 0,
↦ 0, 0, 0, 0, x3,x4,0, 0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9, 0,
↦ 0, 0, 0, 0, x5,x6,0, 0, 0, 0, 0, 0, 7, 0, 0, 8, 0, 0, 9,
↦ 0, 0, 0, 0, 0, 0, x, x2,0, 0, 10,0, 0, 11,0, 0, 12,0, 0,
↦ 0, 0, 0, 0, 0, 0, x3,x4,0, 0, 0, 10,0, 0, 11,0, 0, 12,0,
↦ 0, 0, 0, 0, 0, 0, x5,x6,0, 0, 0, 0, 10,0, 0, 11,0, 0, 12,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x, x2,13,0, 0, 14,0, 0, 15,0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x3,x4,0, 13,0, 0, 14,0, 0, 15,0,
↦ 0, 0, 0, 0, 0, 0, 0, 0, x5,x6,0, 0, 13,0, 0, 14,0, 0, 15
kill m, mm, n, nn, gm, gn;
matrix m[7][3] = x, x-1,x+2, 3x, 4x, x5, x6, x-7, x-8, 9, 10, 11x, 12 -x, 13x, 14x, x
matrix n[2][4] = 1, 2, 3, 4, x, x2, x3, x4;
module mm = m;
module nn = n;
print(mm);
↦ x,    x-1,     x+2,
↦ 3x,   4x,      x5,
↦ x6,   x-7,     x-8,
↦ 9,    10,      11x,
↦ -x+12,13x,     14x,
↦ x15,  x2-8x+16,x17,
↦ 18x,  19x,     20x
print(nn);
↦ 1,2, 3, 4,
↦ x,x2,x3,x4
intmat gm[2][7] = 1, 2, 3, 4, 5, 6, 7, 0, 0, 0, 0, 0, 0, 0;
intmat gn[2][2] = 0, 0, 1, 2;
mm = setModuleGrading(mm, gm);
nn = setModuleGrading(nn, gn);
module mmtnn = multiDegTensor(mm, nn);
↦ // ** redefining mmtnn (module mmtnn = multiDegTensor(mm, nn);) ./example\
   s/multiDegTensor.sing:30
print(mmtnn);
↦ 1,2, 3, 4, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   x,    0,    x-1,     0,       x+2,0,
↦ x,x2,x3,x4,0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   0,    x,    0,       x-1,     0,  x+2,
↦ 0,0, 0, 0, 1,2, 3, 4, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   3x,   0,    4x,      0,       x5, 0,
↦ 0,0, 0, 0, x,x2,x3,x4,0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   0,    3x,   0,       4x,      0,  x5,
↦ 0,0, 0, 0, 0,0, 0, 0, 1,2, 3, 4, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   x6,   0,    x-7,     0,       x-8,0,
↦ 0,0, 0, 0, 0,0, 0, 0, x,x2,x3,x4,0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   0,    x6,   0,       x-7,     0,  x-8,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 1,2, 3, 4, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   9,    0,    10,      0,       11x,0,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, x,x2,x3,x4,0,0, 0, 0, 0,0, 0, 0, 0,0, 0,\
    0,
↦   0,    9,    0,       10,      0,  11x,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 1,2, 3, 4, 0,0, 0, 0, 0,0, 0,\
   0,
↦   -x+12,0,    13x,     0,       14x,0,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, x,x2,x3,x4,0,0, 0, 0, 0,0, 0,\
   0,
↦   0,    -x+12,0,       13x,     0,  14x,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 1,2, 3, 4, 0,0, 0,\
   0,
↦   x15,  0,    x2-8x+16,0,       x17,0,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, x,x2,x3,x4,0,0, 0,\
   0,
↦   0,    x15,  0,       x2-8x+16,0,  x17,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 1,2, 3,\
   4,
↦   18x,  0,    19x,     0,       20x,0,
↦ 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, x,x2,x3\
  ,x4,
↦   0,    18x,  0,       19x,     0,  20x
getModuleGrading(mmtnn);
↦ 1,1,2,2,3,3,4,4,5,5,6,6,7,7,
↦ 1,2,1,2,1,2,1,2,1,2,1,2,1,2
matrix a = mmtnn;
matrix b = tensorMod(mm, nn);
print(a-b);
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

D.15.11.44 multiDegTor

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

D.15.11.45 defineHomogeneous

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: defineHomogeneous(f[, G]); polynomial f, integer matrix G

Purpose: Yields a matrix which has to be appended to the grading group matrix to make the polynomial f homogeneous in the grading by grad.

Example:

```
LIB "multigrading.lib";
ring r =0,(x,y,z),dp;
intmat grad[2][3] =
1,0,1,
0,1,1;
setBaseMultigrading(grad);
poly f = x2y3-z5+x-3zx;
intmat M = defineHomogeneous(f);
M;
↦ 3,0,-1,
↦ 2,-2,-3
defineHomogeneous(f, grad) == M;
↦ 1
isHomogeneous(f);
↦ 0
setBaseMultigrading(grad, M);
isHomogeneous(f);
↦ 1
```

D.15.11.46 pushForward

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: pushForward(f);

Purpose: Computes the finest grading of the image ring which makes the map f a map of graded rings. The group map between the two grading groups is given by transpose((Id, 0)). Pay attention that the group spanned by the columns of the grading group matrix may not be a subgroup of the grading group. Still all columns are needed to find the correct image of the preimage gradings.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
// Setting degrees for preimage ring.;
intmat grad[3][3] =
1,0,0,
0,1,0,
0,0,1;
setBaseMultigrading(grad);
// grading on r:
getVariableWeights();
↦ 1,0,0,
↦ 0,1,0,
↦ 0,0,1
getLattice();
↦ 0,0,0
// only for the purpose of this example
if( voice > 1 ){ /*keepring(r);*/ export(r); }
ring R = 0,(a,b),dp;
ideal i = a2-b2+a6-b5+ab3,a7b+b15-ab6+a6b6;
// The quotient ring by this ideal will become our image ring.;
qring Q = std(i);
listvar();
```

```
↦ // Q                                 [0]  *ring
↦ // R                                 [0]  ring
↦ // grad                              [0]  intmat 3 x 3
↦ // r                                 [0]  ring
map f = r,-a2b6+b5+a3b+a2+ab,-a2b7-3a2b5+b4+a,a6-b6-b3+a2; f;
↦ f[1]=-a2b6+b5+a3b+a2+ab
↦ f[2]=-a2b7-3a2b5+b4+a
↦ f[3]=a6-b6-b3+a2
// TODO: Unfortunately this is not a very spectacular example...:
// Pushing forward f:
pushForward(f);
// due to pushForward we have got new grading on Q
getVariableWeights();
↦ 0,0,
↦ 0,0,
↦ 0,0,
↦ 1,0,
↦ 0,1
getLattice();
↦ 1,0,0,0,0,
↦ 0,1,0,0,0,
↦ 0,0,1,0,0,
↦ 1,1,1,1,0,
↦ 1,1,1,1,1
// only for the purpose of this example
if( voice > 1 ){ kill r; }
```

D.15.11.47 gradiator

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Purpose: coarsens the grading of the basering until the polynom or ideal h becomes homogeneous.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z),dp;
intmat g[2][3] = 1,0,1,0,1,1;
intmat l[2][1] = 3,0;
setBaseMultigrading(g,l);
getLattice();
↦ 3,0
ideal i = -y5+x4,
y6+xz,
x2y;
gradiator(i);
↦ 1
getLattice();
↦ 3,4,2,
↦ 0,-5,-5
isHomogeneous(i);
↦ 1
```

D.15.11.48 hermiteNormalForm

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: hermiteNormalForm(A);

Purpose: Computes the (lower triangular) Hermite Normal Form of the matrix A by column operations.

Return: intmat, the Hermite Normal Form of A

Example:

```
LIB "multigrading.lib";
intmat M[2][5] =
1, 2, 3, 4, 0,
0,10,20,30, 1;
// Hermite Normal Form of M:
print(hermiteNormalForm(M));
↦      1     0     0     0     0
↦      0     1     0     0     0
intmat T[3][4] =
3,3,3,3,
2,1,3,0,
1,2,0,3;
// Hermite Normal Form of T:
print(hermiteNormalForm(T));
↦      3     0     0     0
↦      0     1     0     0
↦      3    -1     0     0
intmat A[4][5] =
1,2,3,2,2,
1,2,3,4,0,
0,5,4,2,1,
3,2,4,0,2;
// Hermite Normal Form of A:
print(hermiteNormalForm(A));
↦      1     0     0     0     0
↦      1     2     0     0     0
↦      0     0     1     0     0
↦      0     1     0     1     0
```

D.15.11.49 smithNormalForm

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: smithNormalForm(A[,opt]); intmat A

Purpose: Computes the Smith Normal Form of A

Return: if no optional argument is given: intmat, the Smith Normal Form of A, otherwise: a list of 3 integer matrices P, D Q, such that D == P*A*Q.

Example:

```
LIB "multigrading.lib";
intmat A[5][7] =
1,0,1,0,-2,9,-71,
0,-24,248,-32,-96,448,-3496,
0,4,-42,4,-8,30,-260,
0,0,0,18,-90,408,-3168,
0,0,0,-32,224,-1008,7872;
```

```
print( smithNormalForm(A) );
↦      1     0     0     0     0     0     0
↦      0     2     2     6    12     0     0
↦      0     0     2     0     0     0     0
↦      0     0     0     8     0     0     0
↦      0     0     0     0    48     0     0
list l = smithNormalForm(A, 5);
l;
↦ [1]:
↦    1,0,0,0,0,
↦    0,0,1,0,0,
↦    0,-1,4,-1,0,
↦    0,3,-12,4,-1,
↦    0,-6,24,-8,3
↦ [2]:
↦    1,0,0,0,0,0,0,
↦    0,2,2,6,12,0,0,
↦    0,0,2,0,0,0,0,
↦    0,0,0,8,0,0,0,
↦    0,0,0,0,48,0,0
↦ [3]:
↦    1,5,45,187,384,8,18591,
↦    0,-52,-461,-1914,-3929,-82,-190555,
↦    0,-5,-53,-218,-445,-10,-23236,
↦    0,0,-28,-109,-215,-7,-16260,
↦    0,0,221,871,1729,53,123084,
↦    0,0,50,197,391,12,27876,
↦    0,0,0,0,0,0,1
l[1]*A*l[3];
↦ 1,0,0,0,0,0,0,
↦ 0,2,2,6,12,0,0,
↦ 0,0,2,0,0,0,0,
↦ 0,0,0,8,0,0,0,
↦ 0,0,0,0,48,0,0
det(l[1]);
↦ 1
det(l[3]);
↦ 1
```

D.15.11.50 hilbertSeries

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Usage: hilbertSeries(I); I is poly/vector/ideal/module

Purpose: computes the multigraded Hilbert Series of I

Note: input must have multigraded-homogeneous generators.
Multigrading should be positive.

Returns: a ring in variables t_(i), s_(i), with polynomials
numerator1 and denominator1 and mutually prime numerator2 and denominator2, quotients of which give the series.

Example:

```
LIB "multigrading.lib";
ring r = 0,(x,y,z,w),dp;
intmat g[2][4]=
1,1,1,1,
0,1,3,4;
setBaseMultigrading(g);
module M = ideal(xw-yz, x2z-y3, xz2-y2w, yw2-z3);
intmat V[2][1]=
1,
0;
M = setModuleGrading(M, V);
def h = hilbertSeries(M); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
factorize(numerator2);
↦ [1]:
↦    _[1]=-1
↦    _[2]=t_(1)
↦    _[3]=t_(1)^3*t_(2)^6-t_(1)^2*t_(2)^6-t_(1)^2*t_(2)^2-t_(1)*t_(2)^3-t_(\
   1)*t_(2)-1
↦ [2]:
↦    1,1,1
factorize(denominator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)-1
↦    _[3]=t_(1)*t_(2)^4-1
↦ [2]:
↦    1,1,1
kill g, h; setring r;
intmat g[2][4]=
1,2,3,4,
0,0,5,8;
setBaseMultigrading(g);
ideal I = x^2, y, z^3;
I = std(I);
list L = multiDegResolution(I, 0, 1);
for( int j = 1; j<=size(L); j++)
{
"----------------------------------- ", j, " -----------------------------";
L[j];
"Module Multigrading: "; print( getModuleGrading(L[j]) );
"Multigrading: "; print(multiDeg(L[j]));
}
↦ -----------------------------------  1  -----------------------------
↦ _[1]=y
↦ _[2]=x2
↦ _[3]=z3
↦ Module Multigrading:
```

```
↦      0
↦      0
↦ Multigrading:
↦      2     2     9
↦      0     0    15
↦ -----------------------------------  2  -----------------------------
↦ _[1]=-x2*gen(1)+y*gen(2)
↦ _[2]=-z3*gen(1)+y*gen(3)
↦ _[3]=-z3*gen(2)+x2*gen(3)
↦ Module Multigrading:
↦      2     2     9
↦      0     0    15
↦ Multigrading:
↦      4    11    11
↦      0    15    15
↦ -----------------------------------  3  -----------------------------
↦ _[1]=z3*gen(1)-x2*gen(2)+y*gen(3)
↦ Module Multigrading:
↦      4    11    11
↦      0    15    15
↦ Multigrading:
↦     13
↦     15
multiDeg(I);
↦ 2,2,9,
↦ 0,0,15
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
factorize(numerator2);
↦ [1]:
↦    _[1]=-1
↦    _[2]=t_(1)+1
↦    _[3]=t_(1)^6*t_(2)^10+t_(1)^3*t_(2)^5+1
↦ [2]:
↦    1,1,1
factorize(denominator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)*t_(2)^2-1
↦    _[3]=t_(1)*t_(2)^2+1
↦    _[4]=t_(1)^2*t_(2)^4+1
↦ [2]:
↦    1,1,1,1
kill r, h, g, V;
////////////////////////////////////////////////
ring R = 0,(x,y,z),dp;
intmat W[2][3] =
1,1, 1,
0,0,-1;
setBaseMultigrading(W);
ideal I = x3y,yz2,y2z,z4;
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
  minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
factorize(numerator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=-t_(1)^5*s_(2)+t_(1)^3*s_(2)^3+t_(1)^2*s_(2)^2+t_(1)^3+t_(1)^2*s_\
  (2)+t_(1)^2+t_(1)*s_(2)+t_(1)+1
↦ [2]:
↦    1,1
factorize(denominator2);
↦ [1]:
↦    _[1]=-1
↦    _[2]=t_(1)-1
↦ [2]:
↦    1,1
kill R, W, h;
////////////////////////////////////////////////
ring R = 0,(x,y,z,a,b,c),dp;
intmat W[2][6] =
1,1, 1,1,1,1,
0,0,-1,0,0,0;
setBaseMultigrading(W);
ideal I = x3y,yz2,y2z,z4;
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
  minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
factorize(numerator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=-t_(1)^5*s_(2)+t_(1)^3*s_(2)^3+t_(1)^2*s_(2)^2+t_(1)^3+t_(1)^2*s_\
  (2)+t_(1)^2+t_(1)*s_(2)+t_(1)+1
↦ [2]:
↦    1,1
factorize(denominator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)-1
↦ [2]:
↦    1,4
kill R, W, h;
////////////////////////////////////////////////
```

```
// This is example 5.3.9. from Robbianos book.
ring R = 0,(x,y,z,w),dp;
intmat W[1][4] =
1,1, 1,1;
setBaseMultigrading(W);
ideal I = z3,y3zw2,x2y4w2xyz2;
hilb(std(I));
↦ (t8-t6-t3+1) / (1-t)^4
↦ (-t7-t6+t2+t+1) / (1-t)^3
↦ // dimension (proj.)  = 2
↦ // degree (proj.)   = 1
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1;
↦ t_(1)^8-t_(1)^6-t_(1)^3+1
denominator1;
↦ t_(1)^4-4*t_(1)^3+6*t_(1)^2-4*t_(1)+1
factorize(numerator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)^7+t_(1)^6-t_(1)^2-t_(1)-1
↦ [2]:
↦    1,1
factorize(denominator2);
↦ [1]:
↦    _[1]=1
↦    _[2]=t_(1)-1
↦ [2]:
↦    1,3
kill h;
////////////////////////////////////////////////
setring R;
ideal I2 = x2,y2,z2; I2;
↦ I2[1]=x2
↦ I2[2]=y2
↦ I2[3]=z2
hilb(std(I2));
↦ (-t6+3t4-3t2+1) / (1-t)^4
↦ (t3+3t2+3t+1) / (1-t)^1
↦ // dimension (proj.)  = 0
↦ // degree (proj.)   = 8
def h = hilbertSeries(I2); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
```

```
numerator1;
↦ -t_(1)^6+3*t_(1)^4-3*t_(1)^2+1
denominator1;
↦ t_(1)^4-4*t_(1)^3+6*t_(1)^2-4*t_(1)+1
kill h;
////////////////////////////////////////////////
setring R;
W = 2,2,2,2;
setBaseMultigrading(W);
getVariableWeights();
↦ 2,2,2,2
intvec w = 2,2,2,2;
hilb(std(I2), 1, w);
↦ 1,0,0,0,-3,0,0,0,3,0,0,0,-1,0
kill w;
def h = hilbertSeries(I2); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)^12+3*t_(1)^8-3*t_(1)^4+1
↦ t_(1)^8-4*t_(1)^6+6*t_(1)^4-4*t_(1)^2+1
kill h;
kill R, W;
////////////////////////////////////////////////
ring R = 0,(x),dp;
intmat W[1][1] =
1;
setBaseMultigrading(W);
ideal I;
I = 1; I;
↦ I[1]=1
hilb(std(I));
↦ (0) / (1-t)^1
↦ (0) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 0
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ 0
↦ -t_(1)+1
kill h;
////////////////////////////////////////////////
setring R;
```

```
I = x; I;
↦ I[1]=x
hilb(std(I));
↦ (-t+1) / (1-t)^1
↦ (1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 1
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
  minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)+1
↦ -t_(1)+1
kill h;
/////////////////////////////////////////////////
setring R;
I = x^5; I;
↦ I[1]=x5
hilb(std(I));
↦ (-t5+1) / (1-t)^1
↦ (t4+t3+t2+t+1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 5
hilb(std(I), 1);
↦ 1,0,0,0,0,-1,0
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
  minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)^5+1
↦ -t_(1)+1
kill h;
/////////////////////////////////////////////////
setring R;
I = x^10; I;
↦ I[1]=x10
hilb(std(I));
↦ (-t10+1) / (1-t)^1
↦ (t9+t8+t7+t6+t5+t4+t3+t2+t+1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 10
def h = hilbertSeries(I); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
  minator1/2'!
```

```
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)^10+1
↦ -t_(1)+1
kill h;
/////////////////////////////////////////////////
setring R;
module M = 1;
M = setModuleGrading(M, W);
hilb(std(M));
↦ (0) / (1-t)^1
↦ (0) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 0
def h = hilbertSeries(M); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ 0
↦ -t_(1)+1
kill h;
/////////////////////////////////////////////////
setring R;
kill M; module M = x^5*gen(1);
//  intmat V[1][3] = 0; // TODO: this would lead to a wrong result!!!?
intmat V[1][1] = 0; // all gen(i) of degree 0!
M = setModuleGrading(M, V);
hilb(std(M));
↦ (-t5+1) / (1-t)^1
↦ (t4+t3+t2+t+1) / (1-t)^0
↦ // dimension (affine) = 0
↦ // degree (affine)  = 5
def h = hilbertSeries(M); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)^5+1
↦ -t_(1)+1
kill h;
/////////////////////////////////////////////////
setring R;
module N = x^5*gen(3);
kill V;
```

```
intmat V[1][3] = 0; // all gen(i) of degree 0!
N = setModuleGrading(N, V);
hilb(std(N));
↦ (-t5+3) / (1-t)^1
↦ (-t5+3) / (1-t)^1
↦ // dimension (proj.)  = 0
↦ // degree (proj.)   = 2
def h = hilbertSeries(N); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -t_(1)^5+3
↦ -t_(1)+1
kill h;
////////////////////////////////////////////////
setring R;
module S = M + N;
S = setModuleGrading(S, V);
hilb(std(S));
↦ (-2t5+3) / (1-t)^1
↦ (-2t5+3) / (1-t)^1
↦ // dimension (proj.)  = 0
↦ // degree (proj.)   = 1
def h = hilbertSeries(S); setring h;
↦  ------------
↦ This proc returns a ring with polynomials called 'numerator1/2' and 'deno\
   minator1/2'!
↦ They represent the first and the second Hilbert Series.
↦ The s_(i)-variables are defined to be the inverse of the t_(i)-variables.
↦  ------------
numerator1; denominator1;
↦ -2*t_(1)^5+3
↦ -t_(1)+1
kill h;
kill V;
kill R, W;
```

D.15.11.51 lll

Procedure from library `multigrading.lib` (see Section D.15.11 [multigrading_lib], page 2458).

Example:

```
LIB "multigrading.lib";
ring R = 0,x,dp;
intmat m[5][5] =
13,25,37,83,294,
12,-33,9,0,64,
77,12,34,6,1,
43,2,88,91,100,
-46,32,37,42,15;
lll(m);
↦ 0,-9,8,43,20,
↦ 12,-33,9,0,64,
↦ -46,41,29,-1,-5,
↦ 77,12,34,6,1,
↦ 23,101,-35,-2,67
list l =
intvec(13,25,37, 83, 294),
intvec(12, -33, 9,0,64),
intvec (77,12,34,6,1),
intvec (43,2,88,91,100),
intvec (-46,32,37,42,15);
lll(l);
↦ [1]:
↦    0,-9,8,43,20
↦ [2]:
↦    12,-33,9,0,64
↦ [3]:
↦    -46,41,29,-1,-5
↦ [4]:
↦    77,12,34,6,1
↦ [5]:
↦    23,101,-35,-2,67
```

D.15.12 pfd_lib

Library: pfd.lib

Purpose: Multivariate Partial Fraction Decomposition

Author: Marcel Wittmann, e-mail: mwittman@mathematik.uni-kl.de

Overview: This Library implements an algorithm based on the work of E. K. Leinartas to write rational functions in multiple variables as a sum of functions with "smaller" numerators and denominators.
This can be used to shorten the IBP reduction coefficients of multi-loop Feynman integrals. For this application,
we also provide a procedure that applies the algorithm to all entries of a matrix of rational functions given as one (possibly very big) txt-file. If you use the library pfd.lib, please cite the corresponding paper [J. Boehm, M. Wittmann, Z. Wu, Y. Xu, Y. Zhang: 'IBP reduction coefficients made simple'] (preprint 2020).

Procedures:

D.15.12.1 pfd

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: pfd(f,g[,debug]); f,g poly, debug int
pfd(f,g[,debug]); f poly, g list, debug int
pfd(arguments[, parallelize]); arguments list, parallelize int

Return: a partial fraction decomposition of f/g as a list `l` where `l[1]` is an ideal generated by irreducible polynomials and `l[2]` is a list of fractions. Each fraction is represented by

a list of
1) the numerator polynomial
2) an intvec of indices `i` for which `l[1][i]` occurs as a factor in the denominator
3) an intvec containing the exponents of those irreducible factors.
Setting `debug` to a positive integer measures runtimes and creates a log file (default: `debug=0`).
The denominator g can also be given in factorized form as a list of an ideal of irreducible non constant polynomials and an intvec of exponents. This can save time since the first step in the algorithm is to factorize g. (A list of the zero-ideal and an empty intvec represents a denominator of 1.)
If instead of f and g, the input is a single list (or even a list of lists) containing elements of the form `list(f,g[,debug])` (`f,g,debug` as above), the algorithm is applied to all entries in parallel (using Section D.2.7 [parallel_lib], page 884), if `parallelize=1` (default) and in sequence if `parallelize=0`. A list (or list of lists) of the results is returned.

Note: The result depends on the monomial ordering. For "small" results use `dp`.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
list dec = pfd(f,g);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
displaypfd_long(dec);   // display result
↦   (2)/((x)*(x-y))
↦ + (-2)/((x-1)*(x-y))
↦ + (-6)/((x)*(x-y)^2)
↦ + (1)/((y)*(x-y)^2)
↦ + (9)/((x-1)*(x-y)^2)
↦ + (2)/((x)^2*(x-y))
↦ (6 terms)
↦
checkpfd(list(f,g),dec);   // check for equality to f/g
↦ 1
// calculate decompositions of a 2x2 matrix of rational functions at once:
list arguments = list(list(f, g),         list(1, f)     ),
list(list(x*y, y+1), list(1, x^2-y^2));
dec = pfd(arguments);
// the result has the same shape as the
// input (2x2 matrix as list of lists):
displaypfd_long(dec[1][1]);
↦   (2)/((x)*(x-y))
↦ + (-2)/((x-1)*(x-y))
↦ + (-6)/((x)*(x-y)^2)
↦ + (1)/((y)*(x-y)^2)
↦ + (9)/((x-1)*(x-y)^2)
↦ + (2)/((x)^2*(x-y))
↦ (6 terms)
↦
displaypfd_long(dec[1][2]);
↦   (1)/((x^3+3*x^2*y-x^2+4*x*y+2*y^2))
↦ (1 terms)
↦
displaypfd_long(dec[2][1]);
↦   (x)
↦ + (-x)/((y+1))
↦ (2 terms)
↦
displaypfd_long(dec[2][2]);
↦   (1)/((x-y)*(x+y))
↦ (1 terms)
↦
// a more complicated example
ring S = 0,(s12,s15,s23,s34,s45),dp;
poly f = 7*s12^4*s15^2 + 11*s12^3*s15^3 + 4*s12^2*s15^4 - 10*s12^4*s15*s23
- 14*s12^3*s15^2*s23 - 4*s12^2*s15^3*s23 + 3*s12^4*s23^2 + 3*s12^3*s15*s23^2
+ 13*s12^4*s15*s34 + 12*s12^3*s15^2*s34 + 2*s12^2*s15^3*s34
- 5*s12^4*s23*s34 + 33*s12^3*s15*s23*s34 + 49*s12^2*s15^2*s23*s34
+ 17*s12*s15^3*s23*s34 - 17*s12^3*s23^2*s34 - 19*s12^2*s15*s23^2*s34
- 5*s12*s15^2*s23^2*s34 - 24*s12^3*s15*s34^2 - 15*s12^2*s15^2*s34^2
+ 2*s12*s15^3*s34^2 + 15*s12^3*s23*s34^2 - 34*s12^2*s15*s23*s34^2
- 31*s12*s15^2*s23*s34^2 + 2*s15^3*s23*s34^2 + 33*s12^2*s23^2*s34^2
+ 29*s12*s15*s23^2*s34^2 + 5*s15^2*s23^2*s34^2 + 9*s12^2*s15*s34^3
- 4*s12*s15^2*s34^3 - 15*s12^2*s23*s34^3 + 9*s12*s15*s23*s34^3
- 4*s15^2*s23*s34^3 - 27*s12*s23^2*s34^3 - 13*s15*s23^2*s34^3
+ 2*s12*s15*s34^4 + 5*s12*s23*s34^4 + 2*s15*s23*s34^4 + 8*s23^2*s34^4
- 6*s12^3*s15^2*s45 - 9*s12^2*s15^3*s45 - 2*s12*s15^4*s45
+ 30*s12^3*s15*s23*s45 + 56*s12^2*s15^2*s23*s45 + 24*s12*s15^3*s23*s45
- 12*s12^3*s23^2*s45 - 23*s12^2*s15*s23^2*s45 - 10*s12*s15^2*s23^2*s45
- 30*s12^3*s15*s34*s45 - 32*s12^2*s15^2*s34*s45 - 6*s12*s15^3*s34*s45
+ 7*s12^3*s23*s34*s45 - 86*s12^2*s15*s23*s34*s45 - 104*s12*s15^2*s23*s34*s45
- 15*s15^3*s23*s34*s45 + 41*s12^2*s23^2*s34*s45 + 51*s12*s15*s23^2*s34*s45
+ 10*s15^2*s23^2*s34*s45 - 5*s12^3*s34^2*s45 + 33*s12^2*s15*s34^2*s45
+ 14*s12*s15^2*s34^2*s45 - 2*s15^3*s34^2*s45 - 21*s12^2*s23*s34^2*s45
+ 62*s12*s15*s23*s34^2*s45 + 28*s15^2*s23*s34^2*s45 - 46*s12*s23^2*s34^2*s45
- 28*s15*s23^2*s34^2*s45 + 10*s12^2*s34^3*s45 - s12*s15*s34^3*s45
+ 4*s15^2*s34^3*s45 + 21*s12*s23*s34^3*s45 - 6*s15*s23*s34^3*s45
+ 17*s23^2*s34^3*s45 - 5*s12*s34^4*s45 - 2*s15*s34^4*s45 - 7*s23*s34^4*s45
- 6*s12^2*s15^2*s45^2 - 5*s12*s15^3*s45^2 - 2*s15^4*s45^2
- 28*s12^2*s15*s23*s45^2 - 42*s12*s15^2*s23*s45^2 - 10*s15^3*s23*s45^2
+ 9*s12^2*s23^2*s45^2 + 10*s12*s15*s23^2*s45^2 + 24*s12^2*s15*s34*s45^2
+ 36*s12*s15^2*s34*s45^2 + 10*s15^3*s34*s45^2 - 11*s12^2*s23*s34*s45^2
+ 31*s12*s15*s23*s34*s45^2 + 25*s15^2*s23*s34*s45^2
- 18*s12*s23^2*s34*s45^2 - 10*s15*s23^2*s34*s45^2 + 4*s12^2*s34^2*s45^2
- 29*s12*s15*s34^2*s45^2 - 17*s15^2*s34^2*s45^2 + 27*s12*s23*s34^2*s45^2
+ 2*s15*s23*s34^2*s45^2 + 9*s23^2*s34^2*s45^2 - 3*s12*s34^3*s45^2
+ 10*s15*s34^3*s45^2 - 16*s23*s34^3*s45^2 - s34^4*s45^2 + 6*s12*s15^2*s45^3
+ 3*s15^3*s45^3 + 8*s12*s15*s23*s45^3 + 10*s15^2*s23*s45^3
- 8*s12*s15*s34*s45^3 - 10*s15^2*s34*s45^3 + 9*s12*s23*s34*s45^3
+ s12*s34^2*s45^3 + 8*s15*s34^2*s45^3 - 9*s23*s34^2*s45^3 - s34^3*s45^3
- s15^2*s45^4 + s15*s34*s45^4;
poly g = 4*s12*s15*(s12 + s15 - s34)*(s15 - s23 - s34)*(s12 + s23 - s45)
*(s12 - s34 - s45)*(s12 + s15 - s34 - s45)*s45;
list dec = pfd(f,g);
↦   (5/4) / (q7*q8)
↦ + (-11/4) / (q6*q8)
↦ + (3/4) / (q5*q8)
↦ + (-11/4) / (q4*q8)
↦ + (1/4) / (q3*q8)
↦ + (5/4) / (q2*q8)
↦ + (1) / (q1*q8)
↦ + (1/2) / (q6*q7)
↦ + (1) / (q5*q6)
↦ + (-3/2) / (q2*q6)
↦ + (-1/4) / (q3*q4)
↦ + (3) / (q1*q4)
↦ + (1/4) / (q1*q2)
↦ + (5/4*s45) / (q1*q7*q8)
↦ + (11/4*s45) / (q1*q6*q8)
↦ + (-3/4*s34) / (q1*q5*q8)
↦ + (11/4*s45) / (q1*q4*q8)
↦ + (-1/4*s34) / (q1*q3*q8)
↦ + (1/4*s45) / (q1*q2*q8)
↦ + (s34) / (q5*q6*q7)
↦ + (-1/2*s45) / (q1*q6*q7)
↦ + (19/4*s34) / (q4*q5*q7)
↦ + (-3/4*s23) / (q2*q5*q6)
↦ + (-7/4*s34) / (q1*q4*q5)
↦ + (2*s34) / (q1*q2*q5)
↦ + (2*s34^2) / (q1*q2*q4*q5)
↦ where
↦ q1 = s12
↦ q2 = s15
↦ q3 = s12+s15-s34
↦ q4 = s15-s23-s34
↦ q5 = s45
↦ q6 = s12+s23-s45
↦ q7 = s12-s34-s45
↦ q8 = s12+s15-s34-s45
↦ (26 terms)
↦
displaypfd(dec);
↦   (5/4) / (q7*q8)
↦ + (-11/4) / (q6*q8)
↦ + (3/4) / (q5*q8)
↦ + (-11/4) / (q4*q8)
↦ + (1/4) / (q3*q8)
↦ + (5/4) / (q2*q8)
↦ + (1) / (q1*q8)
↦ + (1/2) / (q6*q7)
↦ + (1) / (q5*q6)
↦ + (-3/2) / (q2*q6)
↦ + (-1/4) / (q3*q4)
↦ + (3) / (q1*q4)
↦ + (1/4) / (q1*q2)
↦ + (5/4*s45) / (q1*q7*q8)
↦ + (11/4*s45) / (q1*q6*q8)
↦ + (-3/4*s34) / (q1*q5*q8)
↦ + (11/4*s45) / (q1*q4*q8)
↦ + (-1/4*s34) / (q1*q3*q8)
↦ + (1/4*s45) / (q1*q2*q8)
↦ + (s34) / (q5*q6*q7)
↦ + (-1/2*s45) / (q1*q6*q7)
↦ + (19/4*s34) / (q4*q5*q7)
↦ + (-3/4*s23) / (q2*q5*q6)
↦ + (-7/4*s34) / (q1*q4*q5)
↦ + (2*s34) / (q1*q2*q5)
↦ + (2*s34^2) / (q1*q2*q4*q5)
↦ where
↦ q1 = s12
↦ q2 = s15
↦ q3 = s12+s15-s34
↦ q4 = s15-s23-s34
↦ q5 = s45
↦ q6 = s12+s23-s45
↦ q7 = s12-s34-s45
↦ q8 = s12+s15-s34-s45
↦ (26 terms)
↦
checkpfd(list(f,g),dec);
↦ 1
// size comparison:
size(string(f)) + size(string(g));
↦ 4368
size(getStringpfd(dec));
↦ 1055
```

See also: Section D.15.12.2 [checkpfd], page 2521; Section D.15.12.4 [displaypfd], page 2524; Section D.15.12.5 [displaypfd_long], page 2525; Section D.15.12.3 [evaluatepfd], page 2523; Section D.15.12.9 [pfdMat], page 2528.

D.15.12.2 checkpfd

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: checkpfd(list(f,g),dec[,N,C]); f,g poly, dec list, N,C int

Return: 0 or 1

Purpose: test for (mathematical) equality of f/g and a partial fraction decomposition dec. The list dec has to have the same structure as the output of Section D.15.12.1 [pfd],

page 2517.
The denominator g can also be given in factorized form as a list of an ideal of irreducible non constant polynomials and an intvec of exponents. This can save time since the first step in the algorithm is to factorize g. (a list of the zero-ideal and an empty intvec represents a denominator of 1.)
By default the test is done (exactly) by bringing all terms of the decomposition on the same denominator and comparing to f/g.
If additional parameters N [, C] are given and if `N>0`, a probabilistic method is chosen: evaluation at N random points with coordinates between -C and C. This may be faster for big polynomials.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
// partial fraction decomposition of f/g:
list dec = pfd(f,g);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
// some other decomposition (not equal to f/g):
list wrong_dec = pfd(f+1,g);
↦   (1) / (q3*q4)
↦ + (-1) / (q1*q4)
↦ + (-1) / (q2*q3)
↦ + (1) / (q1*q2)
↦ + (-7) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (10) / (q1*q4^2)
↦ + (1) / (q3^2*q4)
↦ + (-1) / (q2*q3^2)
↦ + (1) / (q3*q4^3)
↦ + (-1) / (q2*q4^3)
↦ + (1) / (q3*q4^4)
↦ + (-1) / (q2*q4^4)
↦ + (1) / (q3^2*q4^3)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (14 terms)
```

```
↦
displaypfd_long(dec);
↦   (2)/((x)*(x-y))
↦ + (-2)/((x-1)*(x-y))
↦ + (-6)/((x)*(x-y)^2)
↦ + (1)/((y)*(x-y)^2)
↦ + (9)/((x-1)*(x-y)^2)
↦ + (2)/((x)^2*(x-y))
↦ (6 terms)
↦
list fraction = f,g;
// exact test:
checkpfd(fraction,dec);
↦ 1
checkpfd(fraction,wrong_dec);
↦ 0
// probabilistic test (evaluation at 10 random points):
checkpfd(fraction,dec,10);
↦ 1
checkpfd(fraction,wrong_dec,10);
↦ 0
```

See also: Section D.15.12.1 [pfd], page 2517.

D.15.12.3 evaluatepfd

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: evaluatepfd(dec, values[, mode]); dec list, values ideal, mode int

Return: the number gotten by substituting the numbers generating the ideal `values` for the variables in the partial fraction decomposition `dec`. The list `dec` has to have the same structure as the output of Section D.15.12.1 [pfd], page 2517.
`mode=1`: raise Error in case of division by 0 (default)
`mode=2`: return a second integer which is 1 if the denominator becomes 0, and 0 otherwise.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x+2*y;
poly g = x^2-y^2;
// partial fraction decomposition of f/g:
list dec = pfd(f,g);
↦   (-1/2) / (q2)
↦ + (3/2) / (q1)
↦ where
↦ q1 = x-y
↦ q2 = x+y
↦ (2 terms)
↦
displaypfd_long(dec);
↦   (-1/2)/((x+y))
↦ + (3/2)/((x-y))
↦ (2 terms)
```

```
↦
// evaluation at x=2, y=1:
ideal values = 2,1;
evaluatepfd(dec,values);
↦ 4/3
// compare: f(2,1)/g(2,1) = (2+2*1)/(2^2-1^1) = 4/3
```

See also: Section D.15.12.1 [pfd], page 2517.

D.15.12.4 displaypfd

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: displaypfd(dec); dec list

Purpose: print a partial fraction decomposition `dec` in a readable way. The list `dec` has to have the same structure as the output of Section D.15.12.1 [pfd], page 2517.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
list dec = pfd(f,g);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
displaypfd(dec);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
```

See also: Section D.15.12.5 [displaypfd_long], page 2525; Section D.15.12.6 [getStringpfd], page 2525; Section D.15.12.7 [getStringpfd_indexed], page 2526; Section D.15.12.1 [pfd], page 2517.

D.15.12.5 displaypfd_long

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: displaypfd_long(dec); dec list

Purpose: like Section D.15.12.4 [displaypfd], page 2524, but denominators are written out, not indexed.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
list dec = pfd(f,g);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
displaypfd_long(dec);
↦   (2)/((x)*(x-y))
↦ + (-2)/((x-1)*(x-y))
↦ + (-6)/((x)*(x-y)^2)
↦ + (1)/((y)*(x-y)^2)
↦ + (9)/((x-1)*(x-y)^2)
↦ + (2)/((x)^2*(x-y))
↦ (6 terms)
↦
```

See also: Section D.15.12.4 [displaypfd], page 2524; Section D.15.12.6 [getStringpfd], page 2525; Section D.15.12.7 [getStringpfd_indexed], page 2526; Section D.15.12.1 [pfd], page 2517.

D.15.12.6 getStringpfd

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: getStringpfd(dec); dec list

Purpose: turn a partial fraction decomposition `dec` into one string. The list `dec` has to have the same structure as the output of Section D.15.12.1 [pfd], page 2517.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
list dec = pfd(f,g);
```

```
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
displaypfd_long(dec);
↦   (2)/((x)*(x-y))
↦ + (-2)/((x-1)*(x-y))
↦ + (-6)/((x)*(x-y)^2)
↦ + (1)/((y)*(x-y)^2)
↦ + (9)/((x-1)*(x-y)^2)
↦ + (2)/((x)^2*(x-y))
↦ (6 terms)
↦
getStringpfd(dec);
↦ (2)/((x)*(x-y)) + (-2)/((x-1)*(x-y)) + (-6)/((x)*(x-y)^2) + (1)/((y)*(x-y\
   )^2) + (9)/((x-1)*(x-y)^2) + (2)/((x)^2*(x-y))
```

See also: Section D.15.12.4 [displaypfd], page 2524; Section D.15.12.5 [displaypfd_long], page 2525; Section D.15.12.7 [getStringpfd_indexed], page 2526; Section D.15.12.1 [pfd], page 2517.

D.15.12.7 getStringpfd_indexed

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: getStringpfd_indexed(dec); dec list

Purpose: turn a partial fraction decomposition `dec` into one string, writing the denominator factors just as q1,q2,... . The list `dec` has to have the same structure as the output of Section D.15.12.1 [pfd], page 2517.

Example:

```
LIB "pfd.lib";
ring R = 0,(x,y),dp;
poly f = x^3+3*x^2*y+2*y^2-x^2+4*x*y;
poly g = x^2*y*(x-1)*(x-y)^2;
list dec = pfd(f,g);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
displaypfd(dec);
↦   (2) / (q3*q4)
↦ + (-2) / (q1*q4)
↦ + (-6) / (q3*q4^2)
↦ + (1) / (q2*q4^2)
↦ + (9) / (q1*q4^2)
↦ + (2) / (q3^2*q4)
↦ where
↦ q1 = x-1
↦ q2 = y
↦ q3 = x
↦ q4 = x-y
↦ (6 terms)
↦
getStringpfd_indexed(dec);
↦ (2)/(q3*q4) + (-2)/(q1*q4) + (-6)/(q3*q4^2) + (1)/(q2*q4^2) + (9)/(q1*q4^\
   2) + (2)/(q3^2*q4)
```

See also: Section D.15.12.4 [displaypfd], page 2524; Section D.15.12.5 [displaypfd_long], page 2525; Section D.15.12.6 [getStringpfd], page 2525; Section D.15.12.1 [pfd], page 2517.

D.15.12.8 readInputTXT

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: readInputTXT(file[, mode]), file string, mode int
readInputTXT(filelist[, mode]), filelist list, mode int

Purpose: read matrix of rational functions from a txt-file and turn it into a matrix (i.e. a list of lists) of pairs of polynomials (numerators and denominators). The string `file` should be the [directory +] name of the file in the form `"<path-to-file>/<filename>.txt"`.
The input file should be a list of lists separated by the characters `"{"`, `"}"` and `","`.
Example:
"{{(x+y)/(x^2-x*y), -(x^2*y+1)/(y), x^2}, {(x+1)/y, y/x, 0}}"
Each rational function has to be an expression of the form "a", "(a)/(b)", "(b)^(-n)" or "(a)*(b)^(-n)" where "a","b" stand for polynomials (i.e. strings, that can be parsed as a polynomial with the `execute` command) and "n" stands for a positive integer. A minus sign "-" followed by such an expression is also allowed.
IMPORTANT: The strings "a","b" must NOT contain the symbol "/". (So in case the coefficient field is the rationals, all denominators in the coefficients of numerator and denominator polynomials should be cleared.)
The file should contain less than 2^31 characters (filesize < 2 GB). For bigger files the matrix should be split row-wise into multiple matrices and saved in different files (each smaller than 2 GB). A list of the filenames (in the right order) can then be given as first argument instead.
Also the basering has to match the variable names used in the input file(s).
`mode=1` (default): save result to an ssi-file of the same name
`mode=2`: return result
`mode=3`: save to ssi-file AND return result

See also: Section D.15.12.9 [pfdMat], page 2528.

D.15.12.9 pfdMat

Procedure from library `pfd.lib` (see Section D.15.12 [pfd_lib], page 2517).

Usage: pfdMat(file[, dotest, ignore_nonlin, output_mode, parallelize]); file string, dotest,ignore_nonlin,output_mode,parallelize int

Purpose: apply `pfd` to all entries of a matrix of rational functions saved in a txt-file. The string `file` should be the [directory `+`] name of the file.
The input file can either be a txt-file or an ssi-file created with `readInputTXT`. In case of a txt-file, the base ring has to match and the matrix has to be in the same format specified in Section D.15.12.8 [readInputTXT], page 2527. Also, txt-files that are bigger than 2 GB should be split as described for `readInputTXT` and a list of the filenames can be given as first argument instead.
The result is saved in multiple txt- (and ssi-) files (see below) within the directory of the input file.
Also a logfile is created, which protocols the memory used and the runtimes of `pfd` for each matrix entry in real-time.

There are also 4 optional arguments:
If `dotest` is nonzero, test the results with checkpfd:
`dotest<0` (default): exact test (may be slow),
`dotest>0`: do this amount of probabilistic tests for each entry (see Section D.15.12.2 [checkpfd], page 2521).

If `ignore_nonlin` is nonzero (default), for each denominator, the nonlinear factors in the factorization are removed before applying `pfd` (and added back in in the output files).

If `parallelize` is nonzero (default), the decompositions are calculated in parallel using Section D.2.7 [parallel_lib], page 884.

The parameter `output_mode` controls the output files created:
`output_mode=1` (default): The result consists of two files: `<filename>_pfd_indexed.txt` contains the matrix of all decompositions (as list of lists separated by the characters `"{"`, `"}"` and `","`) where all the denominators are written in factorized form depending on irreducible factors `q1`, `q2`, ... The file `<filename>_denominator_factors.txt` lists all the polynomials `q1`, `q2`, ...
`output_mode=2`: Additionally to mode 1, the file `<filename>_pfd.txt` is created, which also contains the matrix of decompositions but the factors in the denominators are written out.
`output_mode=3`: Additionally to mode 2, the result and some intermediate results are saved as SINGULAR objects in ssi-files:
`<filename>.ssi`: contains the result of `readInputTXT` in case a txt-file was given as input.
`<filename>_factorized_denominators.ssi`: like the first file, but the denominators are saved in factorized form, that is as a list of an ideal of irreducible non constant polynomials and an intvec of exponents.
`<filename>_linear_part.ssi` (only if `ignore_nonlin` is nonzero): like the previous file, but all the irreducible denominator factors are removed
`<filename>_non_linear_factors.ssi` (only if `ignore_nonlin` is nonzero): a list

of an ideal p generated by irreducible polynomials and a matrix (list of lists) of the nonlinear denominator factors of each entry of the input matrix. These are represented as lists of an intvec of indices i for which p[i] occurs as a (nonlinear) factor in the denominator and an intvec containing the exponents of those factors.
<filename>_pfd.ssi: a list, where the first entry is an ideal q of denominator factors and the second entry is a matrix (as list of lists) containing the decompositions, each of which is a list of terms, where a term is represented as in the result of Section D.15.12.1 [pfd], page 2517 by a list containing
1) the numerator polynomial
2) an intvec of indices i for which q[i] occurs as a factor in the denominator
3) an intvec containing the exponents of those irreducible factors.
IMPORTANT: If ignore_nonlin is nonzero, this file contains the decompositions of the entries of the matrix in <filename>_linear_part.ssi. Thus the nonlinear factors, are NOT contained in this file.
output_mode=4: Additionally to mode 3, the direct output of each call of pfd is saved in separate ssi-files called pfd_results_i_j.ssi where i,j are the matrix indices. This creates a lot of files, but may be useful in case the algorithm does not terminate in time for every matrix entry. Other than the files created in mode 1-3, these files are saved in the current directory, rather than the directory of the input file.

See also: Section D.15.12.2 [checkpfd], page 2521; Section D.15.12.10 [checkpfdMat], page 2529; Section D.15.12.1 [pfd], page 2517; Section D.15.12.8 [readInputTXT], page 2527.

D.15.12.10 checkpfdMat

Procedure from library pfd.lib (see Section D.15.12 [pfd_lib], page 2517).

Usage: checkpfdMat(input, output, denomFactors[, N, parallelize]); input,output,denomFactors string, N,parallelize int

Purpose: test the output files of pfdMat for correctness. Input and output (indexed) txt-files have to be given as strings in the form "<path-to-file>/<filename>.txt". The output should be indexed (that is the output file ending in ..._pfd_indexed.txt) and denomFactors has to be the file containing the denominator factors q1, q2, ... (the txt-file ending in ..._denominator_factors.txt).
As for readInputTXT and pfdMat, the basering has to match the variable names used in the input file, which has to be in the same format specified in Section D.15.12.8 [readInputTXT], page 2527. Also, files bigger than 2 GB have to be split as described for readInputTXT and a list of filenames can be given as first argument instead.
If a positive integer N is given, the test is done probabilistically by evaluation at N random points for each entry of the matrix. If N is nonpositive (default), the fractions in the decompositions will be expanded symbolically and compared to the input (may be slower).
If parallelize is nonzero (default), the tests are run in parallel using Section D.2.7 [parallel_lib], page 884.
The result is printed and as in pfdMat a logfile is created showing the results for each matrix entry.

See also: Section D.15.12.2 [checkpfd], page 2521; Section D.15.12.1 [pfd], page 2517; Section D.15.12.9 [pfdMat], page 2528; Section D.15.12.8 [readInputTXT], page 2527.

D.15.13 polyclass_lib

Library: polyclass.lib

Purpose: Data types for normal form equations

Authors: Janko Boehm, email: boehm@mathematik.uni-kl.de
Magdaleen Marais, email: magdaleen.marais@up.ac.za
Gerhard Pfister, email: pfister@mathematik.uni-kl.de

Overview: This library implements a ring independent polynomial type used for the return value in classify2.lib and realclassify.lib. You can use +, * and == for addition, multiplication and comparison. The key over contains the base ring of the polynomial, the key value its value as a polynomial of type poly. The constructor can be called by assigning a polynomial of type poly to a polynomial of type Poly via =.

Moreover the library implements a class NormalformEquation consisting out of a string type, an integer milnorNumber, a Poly normalFormEquation, and integer modality, a list of numbers parameters, a list variables, an integer corank, in the real case, an integer inertiaIndex, a list of open intervals represented as lists consisting out of two rationals used to select a real root of the minimal polynomial (which is stored in the variable minpoly of the polynomial ring containing normalFormEquation, that is, in normalFormEquation.in), or if no minimal polynomial is defined then an interval containing the rational parameter value.

Acknowledgements: This research was supported by
the Staff Exchange Bursary Programme of the University of Pretoria, DFG SPP 1489, DFG TRR 195. The financial assistance of the National Research Foundation (NRF), South Africa, towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the National Research Foundation, South Africa.

Procedures: See also: Section D.6.5 [classify2_lib], page 1654; Section D.6.19 [realclassify_lib], page 1745.

D.15.13.1 makePoly

Procedure from library `polyclass.lib` (see Section D.15.13 [polyclass_lib], page 2529).

Usage: makePoly(f); f poly

Return: make a ring independent Poly from a poly in the basering

Example:

```
LIB "polyclass.lib";
ring R=0,(x,y),dp;
Poly f=3*x^2+x*y+1;
Poly g=2*x+y^3;
f*g;
↦ 3x2y3+xy4+6x3+2x2y+y3+2x
↦
f+g;
↦ y3+3x2+xy+2x+1
↦
f^3;
↦ 27x6+27x5y+9x4y2+x3y3+27x4+18x3y+3x2y2+9x2+3xy+1
↦
```

D.15.13.2 printPoly

Procedure from library `polyclass.lib` (see Section D.15.13 [polyclass_lib], page 2529).

D.15.13.3 printNormalFormEquation

Procedure from library `polyclass.lib` (see Section D.15.13 [polyclass_lib], page 2529).

Usage: printNormalFormEquation(F); F NormalFormEquation

Return: print a normal form equation

Example:

```
LIB "polyclass.lib";
ring R=(0,a),(x,y,z,w),ds;
minpoly = a^2-2;
Poly f=x^4+x^2*y^2+a*y^8+z^2-w^2;
NormalFormEquation F;
F.vars = ringlist(R)[2];
F.realCase = 1;
F.normalFormEquation = f;
F.modality = 1;
F.corank = 2;
F.inertiaIndex = 1;
F.determinacy = 8;
F.milnorNumber = milnor(f.value);
F.parameters = list(list(a*y^8,list(0,2)));
F.singularityType = "X[13]";
F;
↦ Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = X[13]
↦ Normalform equation = z2-w2+x4+x2y2+(a)*y8
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = (a)*y8
↦ Minimal polynomial = (a2-2)
↦ Interval = [0, 2]
↦ Determinacy <= 8
↦
ring R=(0),(x,y,z,w),ds;
↦ // ** redefining R (ring R=(0),(x,y,z,w),ds;) ./examples/printNormalFormE\
   quation.sing:17
Poly f=x^4+x^2*y^2+7*y^8+z^2-w^2;
↦ // ** redefining f (Poly f=x^4+x^2*y^2+7*y^8+z^2-w^2;) ./examples/printNo\
   rmalFormEquation.sing:18
NormalFormEquation F;
↦ // ** redefining F (NormalFormEquation F;) ./examples/printNormalFormEqua\
   tion.sing:19
F.vars = ringlist(R)[2];
F.realCase = 1;
F.normalFormEquation = f;
F.modality = 1;
F.corank = 2;
F.inertiaIndex = 1;
```

```
F.determinacy = 8;
F.milnorNumber = milnor(f.value);
F.parameters = list(list(7*y^8,list(-6,8)));
F.singularityType = "X[13]";
F;
↦ Corank = 2
↦ Inertia index = 1
↦ Normalform equation of type = X[13]
↦ Normalform equation = z2-w2+x4+x2y2+7y8
↦ Milnor number = 13
↦ Modality = 1
↦ Parameter term = 7y8
↦ Determinacy <= 8
↦
```

D.15.14 puiseuxexpansions_lib

Library: puiseuxexpansion.lib

Purpose: Puiseux expansions over algebraic extensions

Authors: J. Boehm, j.boehm at mx.uni-saarland.de
W. Decker, decker at mathematik.uni-kl.de
S. Laplagne, slaplagn at dm.uba.ar
G. Pfister, seelisch at mathematik.uni-kl.de

Overview: This library implements the Newton-Puiseux algorithm to compute Puiseux expansions and provides a class and procedures to work with Puiseux expansions in K[X][Y], where K is the field Q of rational numbers or an algebraic extension of Q.

Procedures:

D.15.14.1 puiseuxList

Procedure from library `puiseuxexpansions.lib` (see Section D.15.14 [puiseuxexpansions_lib], page 2532).

Usage: puiseuxList(PP, maxDeg, iVarX, iVarY); Puiseux expansion PP, int maxDeg is the integer up to which the Puiseux expansions will be computed (if maxDeg = -1 computes the singular part),
int iVarX is the index of the X variable for the ring C{X}[Y], int iVarY, the index of the Y variable for the ring C{X}[Y].

Assume: basering has exactly two variables;
f is convenient, that is, f(x,0) != 0 != f(0,y).

Return: a list with the Puiseux expansions of PP.

Example:

```
LIB "puiseuxexpansions.lib";
ring r=0,(x,y),dp;
Puiseux PP = x5+2x3y-x2y2+3xy5+y6+y7;
puiseuxList(PP, 5, 1, 2);
↦ [1]:
↦    ( 6326x5+634x4+71x3+10x2+3x-1 ) /  1
↦ Denominator of exponent :  1
```

```
↦
↦ [2]:
↦    ( -1876641/256x10-2694181/32768x9-38313/64x8+27513/1024x7-909/16x6+751\
   /128x5-27/4x4+9/8x3-3/2x2+x ) /  1
↦ Denominator of exponent :  2
↦
↦ [3]:
↦    ( -1876641/256x10+2694181/32768x9-38313/64x8-27513/1024x7-909/16x6-751\
   /128x5-27/4x4-9/8x3-3/2x2-x ) /  1
↦ Denominator of exponent :  2
↦
↦ [4]:
↦    ( -746719/256*x^10+(-12549/16*@a)*x^9+13801/64*x^8+(989/16*@a)*x^7-299\
   /16*x^6+(-45/8*@a)*x^5+7/4*x^4+(@a)*x^3-x^2+(@a)*x ) /  1
↦ Denominator of exponent :  2
↦ Minimal polynomial:  (@a^2+1)
↦
↦ [5]:
↦    ( 1813627/128x5+2113/16x4+639/8x3+1/2x2+2x ) /  1
↦ Denominator of exponent :  1
↦
↦ [6]:
↦    ( 5/128x5-1/16x4+1/8x3-1/2x2 ) /  1
↦ Denominator of exponent :  1
↦
```

D.15.14.2 makePuiseux

Procedure from library `puiseuxexpansions.lib` (see Section D.15.14 [puiseuxexpansions_lib], page 2532).

Usage: makePuiseux(f, denom, fr); f polynomial in two variables, denom polynomial in the first variable of the ring, int fr

Return: make a ring independent polynomial over Puiseux series

Example:

```
LIB "puiseuxexpansions.lib";
ring R=0,(x,y),dp;
poly f=3*x^2+x*y+1;
makePuiseux(f,x^2,3);
↦ ( 3x2+xy+1 ) /  x2
↦ Denominator of exponent :  3
↦
```

D.15.14.3 makePuiseuxFromPoly

Procedure from library `puiseuxexpansions.lib` (see Section D.15.14 [puiseuxexpansions_lib], page 2532).

Usage: makePuiseuxFromPoly(f); f polynomial in two variables

Return: make a ring independent polynomial over Puiseux series

Example:

```
LIB "puiseuxexpansions.lib";
ring R=0,(x,y),dp;
poly f=3*x^2+x+1;
makePuiseuxFromPoly(f);
↦ ( 3x2+x+1 ) /  1
↦ Denominator of exponent :  1
↦
```

D.15.14.4 printPuiseux

Procedure from library puiseuxexpansions.lib (see Section D.15.14 [puiseuxexpansions_lib], page 2532).

Usage: printPuiseux(f); f Puiseux expansion

Return: prints information for Puiseux elements

Example:

```
LIB "puiseuxexpansions.lib";
ring R=0,(x,y),dp;
poly f=3*x^2+x*y+1;
Puiseux F = makePuiseux(f,x^2,3);
printPuiseux(F);
↦ ( 3x2+xy+1 ) /  x2
↦ Denominator of exponent :  3
```

D.15.14.5 puiseux

Procedure from library puiseuxexpansions.lib (see Section D.15.14 [puiseuxexpansions_lib], page 2532).

Usage: puiseux(f, maxDeg, atOrigin); f polynomial in two variables, int maxDeg, int atOrigin

Return: the Puiseux expansions of f developed up to degree maxDeg. If atOrigin = 1, only the expansions passing through the origin will be returned.

Example:

```
LIB "puiseuxexpansions.lib";
ring R=0,(x,y),dp;
poly f=y^3 + x^2 + x^8;
puiseux(f,3,0);
↦ [1]:
↦    [1]:
↦       -1/3x20-x2
↦    [2]:
↦       3
↦    [6]:
↦       [1]:
↦          1
↦       [2]:
↦          1
↦    [7]:
↦       [1]:
↦          2
```

D.15.15 ringgb_lib

Library: ringgb.lib

Purpose: Functions for coefficient rings

Author: Oliver Wienand, email: wienand@mathematik.uni-kl.de

Procedures:

D.15.15.1 findZeroPoly

Procedure from library `ringgb.lib` (see Section D.15.15 [ringgb_lib], page 2535).

Usage: findZeroPoly(f); f - a polynomial

Return: zero polynomial with the same leading term as f if exists, otherwise 0

Example:

```
LIB "ringgb.lib";
ring r = (integer, 65536), (y,x), dp;
poly f = 1024*x^8*y^2+11264*x^8*y+28672*x^8+45056*x^7*y^2+36864*x^7*y+16384*x^7+40960
findZeroPoly(f);
↦ 1024y2x8+36864y2x7+64512yx8+2048y2x6+28672yx7+24576y2x5+63488yx6+50176y2x\
   4+40960yx5+53248y2x3+15360yx4+12288y2x2+12288yx3+16384y2x+53248yx2+49152y\
   x
```

D.15.15.2 zeroReduce

Procedure from library `ringgb.lib` (see Section D.15.15 [ringgb_lib], page 2535).

Usage: zeroReduce(f, [i = 0]); f - a polynomial, i - noise level (if != 0 prints all steps)

Return: reduced normal form of f modulo zero polynomials

Example:

```
LIB "ringgb.lib";
ring r = (integer, 65536), (y,x), dp;
poly f = 1024*x^8*y^2+11264*x^8*y+28672*x^8+45056*x^7*y^2+36864*x^7*y+16384*x^7+40960
zeroReduce(f);
↦ 0
kill r;
ring r = (integer, 2, 32), (x,y,z), dp;
// Polynomial 1:
poly p1 = 3795162112*x^3+587202566*x^2*y+2936012853*x*y*z+2281701376*x+548767119*y^3+
// Polynomial 2:
poly p2 = 1647678464*x^3+587202566*x^2*y+2936012853*x*y*z+134217728*x+548767119*y^3+1
zeroReduce(p1-p2);
↦ 0
```

D.15.15.3 testZero

Procedure from library `ringgb.lib` (see Section D.15.15 [ringgb_lib], page 2535).

Usage: testZero(f); f - a polynomial

Return: returns 1 if f is zero as a function and otherwise a counterexample as a list [f(x_1, ..., x_n), x_1, ..., x_n]

Example:

```
LIB "ringgb.lib";
ring r = (integer, 12), (y,x), dp;
poly f = 1024*x^8*y^2+11264*x^8*y+28672*x^8+45056*x^7*y^2+36864*x^7*y+16384*x^7+40960
//zeroReduce(f);
testZero(f);
↦ Teste 144 Belegungen ...
↦ bisher: 11
↦ [1]:
↦    1
↦ [2]:
↦    1
↦ [3]:
↦    4
poly g = findZeroPoly(x2y3);
g;
↦ y3x2+11y3x+9y2x2+3y2x+2yx2+10yx
testZero(g);
↦ Teste 144 Belegungen ...
↦ bisher: 11
↦ bisher: 23
↦ bisher: 35
↦ bisher: 47
↦ bisher: 59
↦ bisher: 71
↦ bisher: 83
↦ bisher: 95
↦ bisher: 107
↦ bisher: 119
↦ bisher: 131
↦ bisher: 143
↦ 1
```

D.15.15.4 noElements

Procedure from library `ringgb.lib` (see Section D.15.15 [ringgb_lib], page 2535).

Usage: noElements(r); r - a ring with a finite coefficient ring of type integer

Return: returns the number of elements of the coefficient ring of r

Example:

```
LIB "ringgb.lib";
ring r = (integer, 233,6), (y,x), dp;
noElements(r);
↦ 160005726539569
```

D.15.16 rwalk_lib

Library: rwalk.lib

Purpose: Groebner Walk Conversion Algorithms

Author: Stephan Oberfranz

Procedures: See also: Section D.4.10 [grwalk_lib], page 1087; Section D.15.20 [swalk_lib], page 2550.

D.15.16.1 prwalk

Procedure from library `rwalk.lib` (see Section D.15.16 [rwalk_lib], page 2536).

Syntax: rwalk(ideal i, int radius);
if size(#)>0 then rwalk(ideal i, int radius, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the Random Perturbation Walk algorithm from the ordering "(a(v),lp)", "dp", "Dp" or "M"
to the ordering "(a(w),lp)", "(a(1,0,...,0),lp)" or "M".

Example:

```
LIB "rwalk.lib";
// compute a Groebner basis of I w.r.t. lp.
ring r = 32003,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
int radius = 1;
int o_perturb_deg = 2;
int t_perturb_deg = 2;
prwalk(I,radius,o_perturb_deg,t_perturb_deg);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.15.16.2 rwalk

Procedure from library `rwalk.lib` (see Section D.15.16 [rwalk_lib], page 2536).

Syntax: rwalk(ideal i, int radius);
if size(#)>0 then rwalk(ideal i, int radius, intvec v, intvec w); intermediate Groebner bases are not reduced if reduction = 0

Type: ideal

Purpose: compute the standard basis of the ideal, calculated via the Random walk algorithm from the ordering
"(a(v),lp)", "dp", "Dp" or "M"
to the ordering "(a(w),lp)", "(a(1,0,...,0),lp)" or "M".

Example:

```
LIB "rwalk.lib";
// compute a Groebner basis of I w.r.t. lp.
ring r = 32003,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
int radius = 1;
int perturb_deg = 2;
```

```
rwalk(I,radius,perturb_deg);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx+8297y8x2+8297y8x+3556y7-8297y6x4+15409y6x3-8297y6x2-8297y5x5+1540\
   9y5x4-8297y5x3+3556y5x2+3556y5x+3556y4x3+3556y4x2-10668y4-10668y3x-8297y2\
   x9-1185y2x8+14224y2x7-1185y2x6-8297y2x5-14223yx7-10666yx6-10666yx5-14223y\
   x4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section D.4.10.1 [fwalk], page 1087; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.15.16.3 frandwalk

Procedure from library `rwalk.lib` (see Section D.15.16 [rwalk_lib], page 2536).

Syntax: frwalk(ideal i, int radius);
frwalk(ideal i, int radius, intvec v, intvec w);

Type: ideal

Purpose: compute the standard basis of the ideal w.r.t. the
lexicographical ordering or a weighted-lex ordering, calculated via the Random Fractal walk algorithm.

Example:

```
LIB "rwalk.lib";
ring r = 0,(z,y,x), lp;
ideal I = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
int reduction = 0;
frandwalk(I,2);
↦ _[1]=y9-y7x2-y7x-y6x3-y6x2-3y6-3y5x-y3x7-3y3x6-3y3x5-y3x4-9y2x5-18y2x4-9y\
   2x3-27yx3-27yx2-27x
↦ _[2]=zx-2/27y8x2-2/27y8x+1/9y7+2/27y6x4+4/27y6x3+2/27y6x2+2/27y5x5+4/27y5\
   x4+2/27y5x3+1/9y5x2+1/9y5x+1/9y4x3+1/9y4x2-1/3y4-1/3y3x+2/27y2x9+8/27y2x8\
   +4/9y2x7+8/27y2x6+2/27y2x5+5/9yx7+5/3yx6+5/3yx5+5/9yx4+x5+2x4+x3
↦ _[3]=zy2+yx2+yx+3
```

See also: Section D.4.10.3 [awalk1], page 1088; Section D.4.10.4 [awalk2], page 1089; Section 5.1.53 [groebner], page 191; Section D.4.10.6 [gwalk], page 1090; Section D.4.10.5 [pwalk], page 1089; Section 5.1.151 [std], page 271; Section 5.1.152 [stdfglm], page 272; Section D.4.10.2 [twalk], page 1088.

D.15.17 sagbigrob_lib

Library: sagbigrob.lib

Purpose: Compute Sagbi-Groebner basis of an ideal of a subalgebra

Authors: Nazish Kanwal, Lecturer of Mathematics, School of Mathematics and Computer Science, Institute of Business Administration, Karachi, Pakistan
Junaid Alam Khan, Associate Professor of Mathematics, School of Mathematics and Computer Science, Institute of Business Administration,
Karachi, Pakistan

Procedures: See also: Section D.4.32 [sagbi_lib], page 1376.

D.15.17.1 LTGS

Procedure from library `sagbigrob.lib` (see Section D.15.17 [sagbigrob_lib], page 2538).

Usage: LTGS(I,A); I ideal of subalgebra A, A subalgebra (which is a finite sagbi basis).

Return: a module M.

Example:

```
LIB "sagbigrob.lib";
// Example 1:
ring r=ZZ,(x,y),Dp;
ideal A=2x2+xy,2y2,3xy;
ideal I=4x2y2+2xy3,18x2y4,36xy5;
LTGS(I,A);
↦ _[1]=9y2*gen(1)-2*gen(2)
↦ _[2]=9y4*gen(1)-xy*gen(3)
↦ _[3]=-xy*gen(3)+2y2*gen(2)
↦ _[4]=xy3*gen(1)
↦ _[5]=-18xy3*gen(1)+2x2*gen(3)+xy*gen(3)
// Example 2:
ring r2=QQ,(x,y),Dp;
ideal A=x2,xy;
ideal I=x3y+x2,x4+x2y2,-x3y3-x2y2;
LTGS(I,A);
↦ _[1]=x2*gen(1)-xy*gen(2)
↦ _[2]=-x2y2*gen(1)-x2*gen(3)
↦ _[3]=-x3y3*gen(2)-x4*gen(3)
```

D.15.17.2 SGNF

Procedure from library `sagbigrob.lib` (see Section D.15.17 [sagbigrob_lib], page 2538).

Usage: SGNF(f,I,A); f polynomial, I ideal of subalgebra
A, A ideal (which is a finite Sagbi bases).

Return: a polynomial h: the normalform of f wrt. I in A

Example:

```
LIB "sagbigrob.lib";
ring r=0,(x,y),Dp;
ideal A=x2,xy;
ideal I=x3y+x2,x4+x2y2,-x3y3-x2y2;
poly f=x6y6+x2;
poly g=x12+x6y6-x8+x5y;
SGNF(f,I,A);
↦ x2y2+x2
SGNF(g,I,A);
↦ 0
```

D.15.17.3 SPOLY

Procedure from library `sagbigrob.lib` (see Section D.15.17 [sagbigrob_lib], page 2538).

Usage: SPOLY(I,A); I ideal of subalgebra A, A subalgebra (which is a finite sagbi basis).

Return: an ideal S: S-polynomials of ideal I

Example:

```
LIB "sagbigrob.lib";
// Example 1:
ring r=ZZ,(x,y),Dp;
ideal A=2x2+xy,2y2,3xy;
ideal I=4x2y2+2xy3,18x2y4;
SPOLY(I,A);
↦ _[1]=18xy5
↦ _[2]=4x3y5+2x2y6
// Example 2:
ring r2=QQ,(x,y),Dp;
ideal A=x2,xy;
ideal I=x3y+x2,x4+x2y2,-x3y3-x2y2;
SPOLY(I,A);
↦ _[1]=-x3y3+x4
↦ _[2]=-x5y5+x6y2
```

D.15.17.4 SGB

Procedure from library `sagbigrob.lib` (see Section D.15.17 [sagbigrob_lib], page 2538).

Usage: SGB(I,A); I ideal of subalgebra A, A subalgebra (which is a finite sagbi basis).

Return: an ideal SB.

Example:

```
LIB "sagbigrob.lib";
// Example 1:
ring r=ZZ,(x,y),Dp;
ideal A=2x2+xy,2y2,3xy;
ideal I=4x2y2+2xy3,18x2y4;
SGB(I,A);
↦ 18xy5
↦ 0
↦ 0
↦ -2x2y6+8xy7
↦ 0
↦ 0
↦ 0
↦ _[1]=4x2y2+2xy3
↦ _[2]=18x2y4
↦ _[3]=18xy5
↦ _[4]=-2x2y6+8xy7
// Example 2:
ring r2=QQ,(w,x,y,z),lp;
ideal A=wxy+2z2, y2-4z, x+3y;
ideal I= wxy-y2+2z2+4z, x+y2+3y-4z, x2+6xy+9y2;
SGB(I,A);
↦ 0
↦ -y6+12y4z-48y2z2+64z3
↦ 0
↦ _[1]=wxy-y2+2z2+4z
↦ _[2]=x+y2+3y-4z
↦ _[3]=x2+6xy+9y2
↦ _[4]=-y6+12y4z-48y2z2+64z3
```

D.15.18 sagbiNormaliz_lib

Library: sagbiNormaliz.lib

Purpose: Provides an interface for the computation of Sagbi bases. It uses normaliz.lib (version 4... or higher) for combinatorial computations.

Authors: Winfried Bruns, wbruns@uos.de

Overview: The library sagbiNormaliz.lib provides functions for the computations of Sagbi bases. It is based on normaliz.lib. Its functions compute Sagbi bases with or without the control by Hilbert functions and/orv degrees. Hilbert functions and degrees require that bthe ambient polynomial ring is standard gtraded. (An extension to general positive gradinmgs would not be difficult.)

Use of this library requires the program Normaliz to be installed and the availability of normaliz.lib. You can download both from @uref{https://github.com/Normaliz/Normaliz/releases}. Please make sure that the executable is in the search path or use setNmzExecPath (defined in normaliz.lib).

The computations of this library require reading Normaliz output files and therefore a file name must be set. The standard file name chosen by the library is NmzSagbiExchange in the current directory. The user can overwrite it by setting another name.

Procedures:

D.15.18.1 sagbiGeneral

Procedure from library `sagbiNormaliz.lib` (see Section D.15.18 [sagbiNormaliz_lib], page 2541).

Usage: sagbiGeneral(ideal Q [, int sagbiMaxRounds, int sorting, int verb]));

Return: An ideal whose generators form the Sagbi basis of the algebra generated by the polynomials in Q as far as computwed in at most sagbiMaxRounds.

If sorting is set, the compouted elements are degrevlex sorted before a round of the algorithm. The optional parameter verb sets the terminal output. Fefault is 1 = on.

The component has a second integer component. Its possible values are: 0, if the full Sagbi basis has not been reached, 1 if this is unknown, and 2 if the full Sagbi basis has been computed.

Example:

```
LIB "sagbiNormaliz.lib";
ring R = 0, (x,y,z),dp;
ideal P = x^6+y^6+z^6, x^7+y^7+z^7;
ideal Q;
int success;
// at most 3 rounds, no sorting, no terminal output
(Q,success) = sagbiGeneral(P,3,0,0);
Q;
↦ Q[1]=x6+y6+z6
↦ Q[2]=x7+y7+z7
↦ Q[3]=x36y6-6/7x35y7+3x30y12-15/7x28y14+5x24y18-20/7x21y21+5x18y24-15/7x14\
   y28+3x12y30-6/7x7y35+x6y36+x36z6+6x30y6z6+15x24y12z6+20x18y18z6+15x12y24z\
   6+6x6y30z6+y36z6-6/7x35z7-30/7x28y7z7-60/7x21y14z7-60/7x14y21z7-30/7x7y28\
   z7-6/7y35z7+3x30z12+15x24y6z12+30x18y12z12+30x12y18z12+15x6y24z12+3y30z12\
   -15/7x28z14-60/7x21y7z14-90/7x14y14z14-60/7x7y21z14-15/7y28z14+5x24z18+20\
   x18y6z18+30x12y12z18+20x6y18z18+5y24z18-20/7x21z21-60/7x14y7z21-60/7x7y14\
   z21-20/7y21z21+5x18z24+15x12y6z24+15x6y12z24+5y18z24-15/7x14z28-30/7x7y7z\
   28-15/7y14z28+3x12z30+6x6y6z30+3y12z30-6/7x7z35-6/7y7z35+x6z36+y6z36
"Note: success = 2 <==> Sagbi basi complete";
↦ Note: success = 2 <==> Sagbi basi complete
"success",success;
↦ success 2
```

D.15.18.2 sagbiByDegree

Procedure from library `sagbiNormaliz.lib` (see Section D.15.18 [sagbiNormaliz_lib], page 2541).

Usage: sagbiByDegree(ideal Q, int sagbiDegreeBound [,int sorting, int verb]);

Return: RETURN: An ideal whose generators form the Sagbi basis of the algebra generated by the polynomials inin degrwees <= Sagbi_degree_bound.

If sorting is set, the compouted elements are degrevlex sorted before a round of the algorithm. The optional parameter verb sets the terminal output. Fefault is 1 = on.

The component has a second integer component. Its possible values are: 0, if the full Sagbi basis has not been reached, 1 if this is unknown, and 2 if the full Sagbi basis has been computed.

Example:

```
LIB "sagbiNormaliz.lib";
ring R = 0, (x,y,z),dp;
ideal P = x^6+y^6+z^6, x^7+y^7+z^7, x^8+y^8+z^8;
ideal Q;
int success;
// degree bound 40, no sorting, no terminal output
(Q,success) = sagbiByDegree(P,40,0,0);
lead(Q);
↦ _[1]=x6
↦ _[2]=x7
↦ _[3]=x8
↦ _[4]=x8y6
↦ _[5]=x18y6
↦ _[6]=x23y7
↦ _[7]=x25y7
"Note: success = 1 <==> Unknown whether complete";
↦ Note: success = 1 <==> Unknown whether complete
"success",success;
↦ success 1
```

D.15.18.3 sagbiHilbControlled

Procedure from library `sagbiNormaliz.lib` (see Section D.15.18 [sagbiNormaliz_lib], page 2541).

Usage: agbiHilbControlled(ideal Q, intvec HS_num_algebra, intvec HS_denom_algebra, int Sagbi_degree_bound [,int finalChweck, int sorting, int verb]);

Return: An ideal whose generators form the Sagbi basis of the algebra generated by the polynomials inin degrwees <= Sagbi_degree_bound. The computatiojn is controlled by the Hilbert series given by the numerator and the denominator.

If sorting is set, the compouted elements are degrevlex sorted before a round of the algorithm. The optional parameter verb sets the terminal output. Fefault is 1 = on.

The component has a second integer component. Its possible values are: 0, if the full Sagbi basis has not been reached, 1 if this is unknown, and 2 if the full Sagbi basis has been computed.

Example:

```
LIB "sagbiNormaliz.lib";
ring R = 0, (x,y,z),dp;
ideal P = x^6+y^6+z^6, x^7+y^7+z^7, x^8+y^8+z^8;
ideal Q;
intvec HS_num = 1;
intvec HS_denom = 6,7,8;
int success;
// degree bound 40, final check, ,no sorting, no terminal output
(Q,success) = sagbiHilbControlled(P,HS_num, HS_denom,40, 1,0,0);
lead(Q);
↦ _[1]=x6
↦ _[2]=x7
↦ _[3]=x8
↦ _[4]=x8y6
↦ _[5]=x18y6
↦ _[6]=x23y7
↦ _[7]=x25y7
"Note: success = 0 <==> Sagbi basis incomplete";
↦ Note: success = 0 <==> Sagbi basis incomplete
"success",success;
↦ success 0
```

D.15.19 stanleyreisner_lib

Library: stanleyreisner.lib

Purpose: Deformations of Stanley-Reiser ideals

Authors:

Overview: Firstly, we implement the graded pieces has certain degree of cotangent modules T1 and T2 for a general Stanley-Reiser ring. And the graded pieces of homomorphisms are represented by lists of integers.

Types: Homomorphism class of homomorphisms

Procedures:

D.15.19.1 T1

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Usage: T1(h); h = ideal.

Assume: I is the ideal generated by the monomials w.r.t. faces.

Return: first order deformation on ideal I.

Theory: The procedure will compute the first order deformaiton.

Example:

```
LIB "stanleyreisner.lib";
ring R=0,(x(1..4)),lp;
ideal i=x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3);
list L=T1(i);
L;
↦ [1]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [2]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [3]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [4]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [5]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [6]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [7]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [8]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [9]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [10]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [11]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
↦ [12]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0
↦ phi.source  ->  phi.over / phi.source
↦ 
```

```
↦ [13]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  1
↦ phi.source  ->  phi.over / phi.source
↦
↦ [14]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  1
↦ phi.source  ->  phi.over / phi.source
↦
↦ [15]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [16]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [17]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [18]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [19]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [20]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [21]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [22]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [23]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [24]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [25]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [26]:
```

```
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [27]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  1
↦ phi.source  ->  phi.over / phi.source
↦
↦ [28]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  1
↦ phi.source  ->  phi.over / phi.source
↦
↦ [29]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [30]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [31]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
↦ [32]:
↦    x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  0,0
↦ phi.source  ->  phi.over / phi.source
↦
```

D.15.19.2 T2

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Usage: T2(h); h = ideal.

Assume: I is the ideal generated by the monomials w.r.t. faces.

Return: second order deformation on ideal I.

Theory: The procedure will compute the second order deformaiton.

Example:

```
LIB "stanleyreisner.lib";
ring R=0,(x(1..4)),lp;
ideal i=x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3);
list L=T2(i);
L;
↦ [1]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [2]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [3]:
↦    phi.relations  ->  phi.over/phi.source
↦
```

```
↦ [4]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [5]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [6]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [7]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [8]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [9]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [10]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [11]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [12]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [13]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [14]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [15]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [16]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [17]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [18]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [19]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [20]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [21]:
↦    phi.relations  ->  phi.over/phi.source
```

```
↦
↦ [22]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [23]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [24]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [25]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [26]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [27]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [28]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [29]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [30]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [31]:
↦    phi.relations  ->  phi.over/phi.source
↦
↦ [32]:
↦    phi.relations  ->  phi.over/phi.source
↦
```

D.15.19.3 makeQPoly

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Usage: makeQPoly(p); p = polynomial.

Assume: p is a polynomial in basering.

Return: a QPoly which is in a quotient ring.

Theory: The procedure will convert a polynomial in basering to a polynomial in a quotient ring.

Example:

```
LIB "stanleyreisner.lib";
ring r=0,(x,y,z),lp;
ideal i=xy;
qring Q=std(i);
poly p=x*y+z;
QPoly q=makeQPoly(p);
q.over;
```

```
↦ // coefficients: QQ
↦ // number of vars : 3
↦ //        block   1 : ordering lp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ // quotient ring from ideal
↦ _[1]=xy
q.value;
↦ z
```

D.15.19.4 fPiece

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Usage: fPiece(I,a,b); I = ideal, a = poly, b = poly.

Assume: I is the ideal generated by the monomials w.r.t. faces, a-b is the degree of the graded piece.

Return: graded piece (of degree a-b) of first order deformation on ideal I.

Theory: The procedure will compute the graded pieces of the first order deformaiton which has degree a-b.

Example:

```
LIB "stanleyreisner.lib";
ring R=0,(x(1..4)),lp;
ideal i=x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3);
FirstOrderDeformation phi=fPiece(i,x(4)^2,x(1)*x(2));
phi;
↦ x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3)  ->  1
↦ phi.source  ->  phi.over / phi.source
↦
```

D.15.19.5 sPiece

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Assume: I is the ideal generated by the monomials w.r.t. faces, a-b is the degree of the graded piece.

Return: graded piece (of degree a-b) of second order deformation on ideal I.

Theory: The procedure will compute the graded pieces of the second order deformaiton which has degree (a-b).

Example:

```
LIB "stanleyreisner.lib";
ring R=0,(x(1..4)),lp;
ideal i=x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3);
SecondOrderDeformation psi=sPiece(i,x(4)^2,x(1)*x(2));
psi;
↦ phi.relations  ->  phi.over/phi.source
↦
```

D.15.19.6 makeLinks

Procedure from library `stanleyreisner.lib` (see Section D.15.19 [stanleyreisner_lib], page 2543).

Usage: makeLinks(I,p,q); I = ideal, p = poly, p=poly.

Assume: I is the ideal generated by the monomials w.r.t. faces.

Return: links of p, graded pieces of first and second order deformation of links on ideal I.

Example:

```
LIB "stanleyreisner.lib";
ring R=0,(x(1..4)),lp;
ideal i=x(1),x(2),x(3),x(4),x(1)*x(4),x(2)*x(4),x(1)*x(3),x(2)*x(3);
LinksDeformation L=makeLinks(i,x(4)^2,x(1)*x(2));
L;
↦ The links of x(4)^2 is :
↦ _[1]=x(1)
↦ _[2]=x(2)
↦ The first order deformation of the links is :
↦ x(1),x(2)  ->  10
↦ phi.source  ->  phi.over / phi.source
↦
↦ The second order deformation of the links is :
↦ phi.relations  ->  phi.over/phi.source
↦
↦
```

D.15.20 swalk_lib

Library: swalk.lib

Purpose: Sagbi Walk Conversion Algorithm

Author: Junaid Alam Khan junaidalamkhan@gmail.com

Overview: A library for computing the Sagbi basis of subalgebra through Sagbi walk algorithm.

Theory: The concept of SAGBI (Subalgebra Analog to Groebner Basis for Ideals) is defined in [L. Robbiano, M. Sweedler: Subalgebra Bases, volume 42, volume 1430 of Lectures Note in Mathematics series, Springer-Verlag (1988),61-87]. The Sagbi Walk algorithm is the subalgebra analogue to the Groebner Walk algorithm which has been proposed in [S. Collart, M. Kalkbrener and D.Mall: Converting bases with the Grobner Walk. J. Symbolic Computation 24 (1997), 465-469].

Procedures: See also: Section D.4.10 [grwalk_lib], page 1087; Section D.15.16 [rwalk_lib], page 2536.

D.15.20.1 swalk

Procedure from library `swalk.lib` (see Section D.15.20 [swalk_lib], page 2550).

Usage: swalk(i[,v,w]); i ideal, v,w int vectors

Return: The sagbi basis of the subalgebra defined by the generators of i, calculated via the Sagbi walk algorithm from the ordering dp to lp if v,w are not given (resp. from the ordering (a(v),lp) to the ordering (a(w),lp) if v and w are given).

Example:

```
LIB "swalk.lib";
ring r = 0,(x,y), lp;
ideal I =x2,y2,xy+y,2xy2+y3;
swalk(I);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=xy+y
↦ _[4]=xy2
↦ _[5]=y3
```

D.15.20.2 rswalk

Procedure from library `swalk.lib` (see Section D.15.20 [swalk_lib], page 2550).

Usage: rswalk(i,weight_rad,p_deg[,v,w]); i ideal, v,w int vectors

Return: The sagbi basis of the subalgebra defined by the generators of i, calculated via the Sagbi walk algorithm from the ordering dp to lp if v,w are not given (resp. from the ordering (a(v),lp) to the ordering (a(w),lp) if v and w are given).

Example:

```
LIB "swalk.lib";
ring r = 0,(x,y), lp;
ideal I =x2,y2,xy+y,2xy2+y3;
rswalk(I,2,2);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=xy+y
↦ _[4]=xy2
↦ _[5]=y3
```

D.15.21 systhreads_lib

Library: systhreads.lib

Purpose: Primitives for Singular's multi-threaded objects

Author: Reimer Behrends

Overview: This library implements basic functionality for shared objects in a multi-threaded system, such as channels, shared tables & lists, and synchronization variables.

D.15.22 tateProdCplxNegGrad_lib

Library: tateProdCplxNegGrad.lib

Purpose: for computing sheaf cohomology on product of projective spaces

Author: Clara Petroll (petroll@mathematik.uni-kl.de)

Overview: In this library, we use Tate resolutions for computing sheaf cohomology of coherent sheaves on products of projective spaces. The algorithms can be used for arbitrary products. We work over the multigraded Cox ring and the corresponding exterior algebra. Multigraded complexes are realized as the newstruct `multigradedcomplex`.

The main algorithm is the one for computing subquotient complexes of a Tate resolution. It allows to compute cohomologytables, respectively hash table of the dimensions of sheaf cohomology groups.

References:
[1] Eisenbud, Erman, Schreyer: Tate Resolutions for Products of Projective Spaces, Acta Mathematica Vietnamica (2015) [2] Eisenbud, Erman, Schreyer: Tate Resolutions on Products of Projective Spaces: Cohomology and Direct Image Complexes (2019)

Procedures:

D.15.22.1 productOfProjectiveSpaces

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: productOfProjectiveSpaces(c); c intvec

Purpose: creates two rings S and E corresponding to the product of projective spaces P^{c_1} x...x P^{c_t}

Assume: input are two integers or an intvec

Return: two rings S,E (homogeneous coordinate ring and the exterior algebra of P^{c_1} x P^{c_2} x...)

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,2;
def (S,E) = productOfProjectiveSpaces(c);
print(S);
↦ polynomial ring, over a field, global ordering
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x(0)(0) x(0)(1) x(1)(0) x(1)(1) x(1)(2)
print(E);
↦ polynomial ring, over a field, global ordering
↦ // coefficients: QQ
↦ // number of vars : 5
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    e(0)(0) e(0)(1) e(1)(0) e(1)(1) e(1)(2)
↦ // noncommutative relations:
↦ //    e(0)(1)e(0)(0)=-e(0)(0)*e(0)(1)
↦ //    e(1)(0)e(0)(0)=-e(0)(0)*e(1)(0)
↦ //    e(1)(1)e(0)(0)=-e(0)(0)*e(1)(1)
↦ //    e(1)(2)e(0)(0)=-e(0)(0)*e(1)(2)
↦ //    e(1)(0)e(0)(1)=-e(0)(1)*e(1)(0)
↦ //    e(1)(1)e(0)(1)=-e(0)(1)*e(1)(1)
↦ //    e(1)(2)e(0)(1)=-e(0)(1)*e(1)(2)
↦ //    e(1)(1)e(1)(0)=-e(1)(0)*e(1)(1)
↦ //    e(1)(2)e(1)(0)=-e(1)(0)*e(1)(2)
↦ //    e(1)(2)e(1)(1)=-e(1)(1)*e(1)(2)
↦ // quotient ring from ideal
↦ _[1]=e(1)(2)^2
↦ _[2]=e(1)(1)^2
↦ _[3]=e(1)(0)^2
↦ _[4]=e(0)(1)^2
↦ _[5]=e(0)(0)^2
intvec d = 2,1,2;
def (S2,E2) = productOfProjectiveSpaces(d);
print(S2);
↦ polynomial ring, over a field, global ordering
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    x(0)(0) x(0)(1) x(0)(2) x(1)(0) x(1)(1) x(\
   2)(0) x(2)(1) x(2)(2)
print(E2);
↦ polynomial ring, over a field, global ordering
↦ // coefficients: QQ
↦ // number of vars : 8
↦ //        block   1 : ordering C
↦ //        block   2 : ordering dp
↦ //                  : names    e(0)(0) e(0)(1) e(0)(2) e(1)(0) e(1)(1) e(\
   2)(0) e(2)(1) e(2)(2)
↦ // noncommutative relations:
↦ //    e(0)(1)e(0)(0)=-e(0)(0)*e(0)(1)
↦ //    e(0)(2)e(0)(0)=-e(0)(0)*e(0)(2)
↦ //    e(1)(0)e(0)(0)=-e(0)(0)*e(1)(0)
↦ //    e(1)(1)e(0)(0)=-e(0)(0)*e(1)(1)
↦ //    e(2)(0)e(0)(0)=-e(0)(0)*e(2)(0)
↦ //    e(2)(1)e(0)(0)=-e(0)(0)*e(2)(1)
↦ //    e(2)(2)e(0)(0)=-e(0)(0)*e(2)(2)
↦ //    e(0)(2)e(0)(1)=-e(0)(1)*e(0)(2)
↦ //    e(1)(0)e(0)(1)=-e(0)(1)*e(1)(0)
↦ //    e(1)(1)e(0)(1)=-e(0)(1)*e(1)(1)
↦ //    e(2)(0)e(0)(1)=-e(0)(1)*e(2)(0)
↦ //    e(2)(1)e(0)(1)=-e(0)(1)*e(2)(1)
↦ //    e(2)(2)e(0)(1)=-e(0)(1)*e(2)(2)
↦ //    e(1)(0)e(0)(2)=-e(0)(2)*e(1)(0)
↦ //    e(1)(1)e(0)(2)=-e(0)(2)*e(1)(1)
↦ //    e(2)(0)e(0)(2)=-e(0)(2)*e(2)(0)
↦ //    e(2)(1)e(0)(2)=-e(0)(2)*e(2)(1)
↦ //    e(2)(2)e(0)(2)=-e(0)(2)*e(2)(2)
↦ //    e(1)(1)e(1)(0)=-e(1)(0)*e(1)(1)
↦ //    e(2)(0)e(1)(0)=-e(1)(0)*e(2)(0)
↦ //    e(2)(1)e(1)(0)=-e(1)(0)*e(2)(1)
↦ //    e(2)(2)e(1)(0)=-e(1)(0)*e(2)(2)
↦ //    e(2)(0)e(1)(1)=-e(1)(1)*e(2)(0)
↦ //    e(2)(1)e(1)(1)=-e(1)(1)*e(2)(1)
↦ //    e(2)(2)e(1)(1)=-e(1)(1)*e(2)(2)
↦ //    e(2)(1)e(2)(0)=-e(2)(0)*e(2)(1)
↦ //    e(2)(2)e(2)(0)=-e(2)(0)*e(2)(2)
↦ //    e(2)(2)e(2)(1)=-e(2)(1)*e(2)(2)
↦ // quotient ring from ideal
↦ _[1]=e(2)(2)^2
↦ _[2]=e(2)(1)^2
↦ _[3]=e(2)(0)^2
↦ _[4]=e(1)(1)^2
↦ _[5]=e(1)(0)^2
↦ _[6]=e(0)(2)^2
↦ _[7]=e(0)(1)^2
↦ _[8]=e(0)(0)^2
```

D.15.22.2 truncateM

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: truncateM(M,c); M module, c intvec

Purpose: truncate M at c

Assume: `M` is multigraded S-module with S multigraded ring, c is an intvec of the right length

Return: module, the truncated module M_{>= c}

Note: Output is the truncated module (multigraded , grading is not shifted), works for arbitrary products

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,1,1;
def(S,E) = productOfProjectiveSpaces(c);
setring(S);
intmat grading[3][2] = 0,0,0,0,0,0;
module te = freemodule(2);
te = setModuleGrading(te,grading);
intvec c = 1,1,1;
↦ // ** redefining c (intvec c = 1,1,1;) ./examples/truncateM.sing:8
module Mtrunc = truncateM(te,c);
Mtrunc;
↦ Mtrunc[1]=x(0)(1)*x(1)(1)*x(2)(1)*gen(1)
↦ Mtrunc[2]=x(0)(0)*x(1)(1)*x(2)(1)*gen(1)
↦ Mtrunc[3]=x(0)(1)*x(1)(0)*x(2)(1)*gen(1)
↦ Mtrunc[4]=x(0)(0)*x(1)(0)*x(2)(1)*gen(1)
↦ Mtrunc[5]=x(0)(1)*x(1)(1)*x(2)(0)*gen(1)
↦ Mtrunc[6]=x(0)(0)*x(1)(1)*x(2)(0)*gen(1)
↦ Mtrunc[7]=x(0)(1)*x(1)(0)*x(2)(0)*gen(1)
↦ Mtrunc[8]=x(0)(0)*x(1)(0)*x(2)(0)*gen(1)
↦ Mtrunc[9]=x(0)(1)*x(1)(1)*x(2)(1)*gen(2)
↦ Mtrunc[10]=x(0)(0)*x(1)(1)*x(2)(1)*gen(2)
↦ Mtrunc[11]=x(0)(1)*x(1)(0)*x(2)(1)*gen(2)
↦ Mtrunc[12]=x(0)(0)*x(1)(0)*x(2)(1)*gen(2)
↦ Mtrunc[13]=x(0)(1)*x(1)(1)*x(2)(0)*gen(2)
↦ Mtrunc[14]=x(0)(0)*x(1)(1)*x(2)(0)*gen(2)
↦ Mtrunc[15]=x(0)(1)*x(1)(0)*x(2)(0)*gen(2)
↦ Mtrunc[16]=x(0)(0)*x(1)(0)*x(2)(0)*gen(2)
getModuleGrading(Mtrunc);
↦ 0,0,
↦ 0,0,
↦ 0,0
multiDeg(Mtrunc);
↦ 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
↦ 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
↦ 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

D.15.22.3 truncateCoker

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: truncateCoker(M,c); M module, c intvec

Purpose: truncate cokernel coker(M) at the multidegree c

Return: module, which is a presentation matrix of the truncation of coker(M) at c

Example:

```
LIB "tateProdCplxNegGrad.lib";
// example 1
intvec c1 = 1,1,1;
def(S1,E1) = productOfProjectiveSpaces(c1);
setring(S1);
module M1= 0;
intmat grading1[3][1] = 0,0,0;
M1 = setModuleGrading(M1,grading1);
truncateCoker(M1,c1);
↦ _[1]=x(0)(1)*gen(2)-x(0)(0)*gen(1)
↦ _[2]=x(1)(1)*gen(3)-x(1)(0)*gen(1)
↦ _[3]=x(1)(1)*gen(4)-x(1)(0)*gen(2)
↦ _[4]=x(0)(1)*gen(4)-x(0)(0)*gen(3)
↦ _[5]=x(2)(1)*gen(5)-x(2)(0)*gen(1)
↦ _[6]=x(2)(1)*gen(6)-x(2)(0)*gen(2)
↦ _[7]=x(0)(1)*gen(6)-x(0)(0)*gen(5)
↦ _[8]=x(2)(1)*gen(7)-x(2)(0)*gen(3)
↦ _[9]=x(1)(1)*gen(7)-x(1)(0)*gen(5)
↦ _[10]=x(2)(1)*gen(8)-x(2)(0)*gen(4)
↦ _[11]=x(1)(1)*gen(8)-x(1)(0)*gen(6)
↦ _[12]=x(0)(1)*gen(8)-x(0)(0)*gen(7)
// example 2
intvec c2 = 1,1;
def (S2,E2) = productOfProjectiveSpaces(c2);
setring(S2);
module M2 = 0;
intmat grading2[2][1] = 0,0;
M2 = setModuleGrading(M2,grading2);
truncateCoker(M2,c2);
↦ _[1]=x(0)(1)*gen(2)-x(0)(0)*gen(1)
↦ _[2]=x(1)(1)*gen(3)-x(1)(0)*gen(1)
↦ _[3]=x(1)(1)*gen(4)-x(1)(0)*gen(2)
↦ _[4]=x(0)(1)*gen(4)-x(0)(0)*gen(3)
```

D.15.22.4 symExt

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: symExt(m); m matrix

Purpose: computes differential R(M_0) -> R(M_1) for the module M over S corresponding to the linear presentation matrix m, however, in order to get the result, m has to be fetched to the exterior algebra E

Assume: m a matrix, linear presentation matrix over S; Note: also works for nonlinear matrices, but makes no sense to use it in this case

Return: matrix B representing R(M_0) -> R(M_1)

Note: output lives in S (not as in Macaulay2 in the ring E, to get the same result, just fetch the matrix to E)

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,2;
def (S,E) = productOfProjectiveSpaces(c);
setring(S);
matrix m[4][2] = x(0)(0), x(1)(0),x(0)(1),0,0,x(1)(1), 0,x(1)(2);
matrix A = symExt(m);
print(A);
↦ 0,       x(0)(0),0,       0,
↦ 0,       0,      x(0)(0), 0,
↦ 0,       0,      0,       x(0)(0),
↦ x(0)(1), 0,      0,       0,
↦ -x(0)(0),x(0)(1),0,       0,
↦ 0,       0,      x(0)(1), 0,
↦ 0,       0,      0,       x(0)(1),
↦ 0,       x(1)(0),0,       0,
↦ 0,       0,      x(1)(0), 0,
↦ 0,       0,      0,       x(1)(0),
↦ x(1)(1), 0,      0,       0,
↦ 0,       x(1)(1),0,       0,
↦ -x(1)(0),0,      x(1)(1), 0,
↦ 0,       0,      0,       x(1)(1),
↦ x(1)(2), 0,      0,       0,
↦ 0,       x(1)(2),0,       0,
↦ 0,       0,      x(1)(2), 0,
↦ 0,       0,      -x(1)(1),x(1)(2)
setring(E);
print(fetch(S,A));
↦ 0,       e(0)(0),0,       0,
↦ 0,       0,      e(0)(0), 0,
↦ 0,       0,      0,       e(0)(0),
↦ e(0)(1), 0,      0,       0,
↦ -e(0)(0),e(0)(1),0,       0,
↦ 0,       0,      e(0)(1), 0,
↦ 0,       0,      0,       e(0)(1),
↦ 0,       e(1)(0),0,       0,
↦ 0,       0,      e(1)(0), 0,
↦ 0,       0,      0,       e(1)(0),
↦ e(1)(1), 0,      0,       0,
↦ 0,       e(1)(1),0,       0,
↦ -e(1)(0),0,      e(1)(1), 0,
↦ 0,       0,      0,       e(1)(1),
↦ e(1)(2), 0,      0,       0,
↦ 0,       e(1)(2),0,       0,
↦ 0,       0,      e(1)(2), 0,
↦ 0,       0,      -e(1)(1),e(1)(2)
```

D.15.22.5 sufficientlyPositiveMultidegree

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: sufficientlyPositiveMultidegree(M); M module

Purpose: computes a sufficiently positive multidegree for coker(M)

Assume: M is multigraded S-module

Return: intvec that is sufficiently positive for M

Example:

```
LIB "tateProdCplxNegGrad.lib";
// example 1
intvec c1 = 1,2;
def (S1,E1) = productOfProjectiveSpaces(c1);
setring(S1);
module M1 = x(0)(0),x(1)(0)^3 + x(1)(1)^3 +x(1)(2)^3;
intmat grading1[2][1] = 0,0;
M1 = setModuleGrading(M1,grading1);
sufficientlyPositiveMultidegree(M1);
↦ 1,3
// example 2
intvec c2 = 1,1;
def (S2,E2) = productOfProjectiveSpaces(c2);
setring(S2);
intmat grading2[2][1] = -1,-1;
module M2 = 0;
M2 = setModuleGrading(M2,grading2);
sufficientlyPositiveMultidegree(M2);
↦ -1,-1
// example 3
intvec c3 = 1,1,1;
def (S3,E3) = productOfProjectiveSpaces(c3);
setring(S3);
module M3 = 0;
intmat grading3[3][1] = -1,-1,-1;
M3 = setModuleGrading(M3,grading3);
sufficientlyPositiveMultidegree(M3);
↦ -1,-1,-1
```

D.15.22.6 tateResolution

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: tateResolution(M,low,high); M module, L list, low intvec, high intvec

Purpose: compute tate resolution of coker(M) where M is Z^t-graded S-module

Assume: M a module over multigraded ring S

Return: (E,tate), tate a multigradedcomplex, E the ring in which tate has to be viewed, however note that tate is not ring dependent

Example:

```
LIB "tateProdCplxNegGrad.lib";
// example 1
intvec c1 = 1,1,1;
intvec low1 = 0,0,0;
intvec high1 = 0,1,0;
def (S1,E1) = productOfProjectiveSpaces(c1);
setring(S1);
module M1 = 0;
intmat grading1[3][1] = -1,-1,-1;
M1 = setModuleGrading(M1,grading1);
multigradedcomplex tate1;
(E1,tate1) = tateResolution(M1,low1,high1);
setring(E1);
tate1;
↦ E^12  <--  E^20
↦ -1         0
↦
tate1.differentials;
↦ [1]:
↦    _[1]=-e(0)(0)*gen(2)-e(0)(1)*gen(1)
↦    _[2]=-e(1)(0)*gen(3)-e(1)(1)*gen(1)
↦    _[3]=-e(1)(0)*gen(4)-e(1)(1)*gen(2)
↦    _[4]=-e(0)(0)*gen(4)-e(0)(1)*gen(3)
↦    _[5]=-e(1)(0)*gen(5)-e(1)(1)*gen(3)
↦    _[6]=-e(1)(0)*gen(6)-e(1)(1)*gen(4)
↦    _[7]=-e(0)(0)*gen(6)-e(0)(1)*gen(5)
↦    _[8]=-e(2)(0)*gen(7)-e(2)(1)*gen(1)
↦    _[9]=-e(2)(0)*gen(8)-e(2)(1)*gen(2)
↦    _[10]=-e(0)(0)*gen(8)-e(0)(1)*gen(7)
↦    _[11]=-e(2)(0)*gen(9)-e(2)(1)*gen(3)
↦    _[12]=-e(1)(0)*gen(9)-e(1)(1)*gen(7)
↦    _[13]=-e(2)(0)*gen(10)-e(2)(1)*gen(4)
↦    _[14]=-e(1)(0)*gen(10)-e(1)(1)*gen(8)
↦    _[15]=-e(0)(0)*gen(10)-e(0)(1)*gen(9)
↦    _[16]=-e(2)(0)*gen(11)-e(2)(1)*gen(5)
↦    _[17]=-e(1)(0)*gen(11)-e(1)(1)*gen(9)
↦    _[18]=-e(2)(0)*gen(12)-e(2)(1)*gen(6)
↦    _[19]=-e(1)(0)*gen(12)-e(1)(1)*gen(10)
↦    _[20]=-e(0)(0)*gen(12)-e(0)(1)*gen(11)
// example 2
intvec c2 = 1,2;
def (S2,E2) = productOfProjectiveSpaces(c2);
setring(S2);
intvec low2 = -3,-3;
intvec high2 = 0,0;
module M2 = x(0)(0),x(1)(0)^3 + x(1)(1)^3 +x(1)(2)^3;;
intmat grading2[2][1] = 0,0;
M2 = setModuleGrading(M2,grading2);
multigradedcomplex tate2;
```

```
(E2,tate2) = tateResolution(M2,low2,high2);
setring(E2);
tate2;
↦ E^6  <--  E^10  <--  E^14  <--  E^20  <--  E^29  <--  E^41  <--  E^56  <-\
  -  E^74  <--  E^95
↦ -2        -1         0          1          2          3          4       \
     5          6
↦
// example 3
intvec c3 = 1,1;
def (S3,E3) = productOfProjectiveSpaces(c3);
intvec low3 = -3,-3;
intvec high3 = 3,3;
setring(S3);
module M3 = 0;
intmat grading3[2][1] = -1,-1;
M3 = setModuleGrading(M3,grading3);
multigradedcomplex tate3;
(E3,tate3) = tateResolution(M3,low3,high3);
setring(E3);
tate3;
↦ E^25  <--  E^40  <--  E^46  <--  E^44  <--  E^35  <--  E^30  <--  E^38  <\
  --  E^56  <--  E^81  <--  E^110  <--  E^141  <--  E^174  <--  E^210
↦ -6         -5         -4         -3         -2         -1         0      \
      1          2          3           4           5           6
↦
```

D.15.22.7 cohomologyMatrix

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: cohomologyMatrix(M,L,low,high); M module, L list, low intvec, high intvec

Purpose: computes the cohomology matrix of the sheaf corresponding to coker(M)

Assume: M module over S, L list of two rings S and E (e.g. result of productOfProjectiveSpaces) first entry L[1] = S and L[2] = E, integer vectors low <= high

Return: ring Z in which cohomology matrix lives, it is exported in the variable cohomologymat, cohomologymat covers all cohomology groups of twists in the range between low and high

Example:

```
LIB "tateProdCplxNegGrad.lib";
// example 1
intvec c1 = 1,1;
def (S1,E1) = productOfProjectiveSpaces(c1);
intvec low1 = -3,-3;
intvec high1 = 3,3;
setring(S1);
module M1 = 0;
intmat grading1[2][1] = -1,-1;
M1 = setModuleGrading(M1,grading1);
ring Z1 = cohomologyMatrix(M1,low1,high1);
```

```
setring(Z1);
print(cohomologymat);
↦ 5h,0,5,10,15,20,25,
↦ 4h,0,4,8, 12,16,20,
↦ 3h,0,3,6, 9, 12,15,
↦ 2h,0,2,4, 6, 8, 10,
↦ h, 0,1,2, 3, 4, 5,
↦ 0, 0,0,0, 0, 0, 0,
↦ h2,0,h,2h,3h,4h,5h
// example 2
intvec c2 = 1,2;
def (S2,E2) = productOfProjectiveSpaces(c2);
intvec low2 = -3,-3;
intvec high2 = 0,0;
setring(S2);
module M2 = 0;
intmat grading2[2][1] = 0,0;
M2 = setModuleGrading(M2,grading2);
ring Z2 = cohomologyMatrix(M2,low2,high2);
setring(Z2);
print(cohomologymat);
↦ 2h, h, 0,1,
↦ 0,  0, 0,0,
↦ 0,  0, 0,0,
↦ 2h3,h3,0,h2
// example 3
setring(S2);
module M3 = x(0)(0),x(1)(0)^3 + x(1)(1)^3 +x(1)(2)^3;
intmat grading3[2][1] = 0,0;
M3 = setModuleGrading(M3,grading3);
ring Z3 = cohomologyMatrix(M3,low2,high2);
setring(Z3);
print(cohomologymat);
↦ h+1,h+1,h+1,h+1,
↦ 3h, 3h, 3h, 3h,
↦ 6h, 6h, 6h, 6h,
↦ 9h, 9h, 9h, 9h
```

D.15.22.8 cohomologyMatrixFromResolution

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: cohomologyMatrixFromResolution(T,low,high); T multigradedcomplex, low intvec, high intvec

Purpose: computes the cohomology matrix corresponding to the multigraded complex T (part of a Tate resolution)

Assume: T is a multigraded complex representing a part of a Tate resolution (for example output of tateResolution), basering is E

Return: ring Z in which cohomology matrix lives, it is exported in the variable cohomologymat, cohomologymat stores information in the range between low and high

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,1;
def (S,E) = productOfProjectiveSpaces(c);
intvec low = -3,-3;
intvec high = 3,3;
setring(S);
module M = 0;
intmat grading[2][1] = -1,-1;
M = setModuleGrading(M,grading);
multigradedcomplex tate;
(E,tate) = tateResolution(M,low,high);
setring(E);
ring Z = cohomologyMatrixFromResolution(tate,low,high);
setring(Z);
print(cohomologymat);
↦ 5h,0,5,10,15,20,25,
↦ 4h,0,4,8, 12,16,20,
↦ 3h,0,3,6, 9, 12,15,
↦ 2h,0,2,4, 6, 8, 10,
↦ h, 0,1,2, 3, 4, 5,
↦ 0, 0,0,0, 0, 0, 0,
↦ h2,0,h,2h,3h,4h,5h
```

D.15.22.9 eulerPolynomialTable

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: eulerPolynomialTable(M,low,high); M module, L list, low intvec, high intvec

Purpose: computes hash table of euler polynomials of twists of coker(M) in the range between low and high

Assume: M module, note that at the moment M is a module over S,

Return: (Z,eulerpolynomialtable), where eulerpolynomialtable is a hash table with entries in the ring Z = ZZ[h]

Note: this function works for arbitrary products P^{n_1} x \cdots x P^{n_t} and corresponding Z^t-gradings, entries can be accessed via eulerpolynomialtable*(a_1,...,a_t) where a=(a_1,...,a_t) is a multidegree betweeen low and high

Example:

```
LIB "tateProdCplxNegGrad.lib";
// example 1
intvec c1 = 1,1;
def (S1,E1) = productOfProjectiveSpaces(c1);
intvec low1 = -3,-3;
intvec high1 = 3,3;
setring(S1);
module M1 = 0;
intmat grading1[2][1] = -1,-1;
M1 = setModuleGrading(M1,grading1);
def (Z1,eulerTable1) = eulerPolynomialTable(M1,low1,high1);
setring(Z1);
print(eulerTable1);
```

```
↦ -3,-3 => h2
↦ -2,-3 => 0
↦ -1,-3 => h
↦ 0,-3 => 2h
↦ 1,-3 => 3h
↦ 2,-3 => 4h
↦ 3,-3 => 5h
↦ -3,-2 => 0
↦ -3,-1 => h
↦ -3,0 => 2h
↦ -3,1 => 3h
↦ -3,2 => 4h
↦ -3,3 => 5h
↦ -2,-2 => 0
↦ -2,-1 => 0
↦ -2,0 => 0
↦ -2,1 => 0
↦ -2,2 => 0
↦ -2,3 => 0
↦ -1,-2 => 0
↦ -1,-1 => 1
↦ -1,0 => 2
↦ -1,1 => 3
↦ -1,2 => 4
↦ -1,3 => 5
↦ 0,-2 => 0
↦ 0,-1 => 2
↦ 0,0 => 4
↦ 0,1 => 6
↦ 0,2 => 8
↦ 0,3 => 10
↦ 1,-2 => 0
↦ 1,-1 => 3
↦ 1,0 => 6
↦ 1,1 => 9
↦ 1,2 => 12
↦ 1,3 => 15
↦ 2,-2 => 0
↦ 2,-1 => 4
↦ 2,0 => 8
↦ 2,1 => 12
↦ 2,2 => 16
↦ 2,3 => 20
↦ 3,-2 => 0
↦ 3,-1 => 5
↦ 3,0 => 10
↦ 3,1 => 15
↦ 3,2 => 20
↦ 3,3 => 25
eulerTable1*low1;
↦ h2
setring(S1);
ring Z = cohomologyMatrix(M1,low1,high1);
```

```
setring(Z);
print(cohomologymat);
↦ 5h,0,5,10,15,20,25,
↦ 4h,0,4,8, 12,16,20,
↦ 3h,0,3,6, 9, 12,15,
↦ 2h,0,2,4, 6, 8, 10,
↦ h, 0,1,2, 3, 4, 5,
↦ 0, 0,0,0, 0, 0, 0,
↦ h2,0,h,2h,3h,4h,5h
// example 2
intvec c2 =  1,1,1;
def (S2,E2) = productOfProjectiveSpaces(c2);
setring(S2);
intvec low2 = 0,0,0;
intvec high2 = 1,1,1;
module M2 = 0;
intmat grading2[3][1] = -1,-1,-1;
M2 = setModuleGrading(M2,grading2);
def (Z2,eulerTable2) = eulerPolynomialTable(M2,low2,high2);
setring(Z2);
print(eulerTable2);
↦ 0,0,0 => 8
↦ 1,0,0 => 12
↦ 0,1,0 => 12
↦ 1,1,0 => 18
↦ 0,0,1 => 12
↦ 1,0,1 => 18
↦ 0,1,1 => 18
↦ 1,1,1 => 27
```

D.15.22.10 cohomologyHashTable

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: cohomologyHashTable(M,L,low,high); M module, low intvec, high intvec

Purpose: computes hashtable of sheaf cohomology groups of twists in the range between between low and high corresponding to coker(M)

Assume: M module representing a sheaf F on the product of t projective spaces, note that at the moment M is a module over S,

Return: cohomologytable where cohomologytable is a hash table with integer vectors in ZZ^{t+1} as keys, entries can be accessed via cohomologytable*(c_1,...,c_t,i) = dim(H^i(F(c_1,...,c_t)))

Note: this function works for arbitrary products P^{n_1} x \cdots x P^{n_t} and corresponding Z^t-gradings

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,1;
def (S,E) = productOfProjectiveSpaces(c);
intvec low = -3,-3;
intvec high = 3,3;
setring(S);
module M = 0;
intmat grading[2][1] = -1,-1;
M = setModuleGrading(M,grading);
def cohomologytable = cohomologyHashTable(M,low,high);
print(cohomologytable);
↦ -3,-3,0 => 0
↦ -3,-3,1 => 0
↦ -3,-3,2 => 1
↦ -2,-3,0 => 0
↦ -2,-3,1 => 0
↦ -2,-3,2 => 0
↦ -1,-3,0 => 0
↦ -1,-3,1 => 1
↦ -1,-3,2 => 0
↦ 0,-3,0 => 0
↦ 0,-3,1 => 2
↦ 0,-3,2 => 0
↦ 1,-3,0 => 0
↦ 1,-3,1 => 3
↦ 1,-3,2 => 0
↦ 2,-3,0 => 0
↦ 2,-3,1 => 4
↦ 2,-3,2 => 0
↦ 3,-3,0 => 0
↦ 3,-3,1 => 5
↦ 3,-3,2 => 0
↦ -3,-2,0 => 0
↦ -3,-2,1 => 0
↦ -3,-2,2 => 0
↦ -3,-1,0 => 0
↦ -3,-1,1 => 1
↦ -3,-1,2 => 0
↦ -3,0,0 => 0
↦ -3,0,1 => 2
↦ -3,0,2 => 0
↦ -3,1,0 => 0
↦ -3,1,1 => 3
↦ -3,1,2 => 0
↦ -3,2,0 => 0
↦ -3,2,1 => 4
↦ -3,2,2 => 0
↦ -3,3,0 => 0
↦ -3,3,1 => 5
↦ -3,3,2 => 0
↦ -2,-2,0 => 0
↦ -2,-2,1 => 0
↦ -2,-2,2 => 0
↦ -2,-1,0 => 0
↦ -2,-1,1 => 0
↦ -2,-1,2 => 0
↦ -2,0,0 => 0
↦ -2,0,1 => 0
↦ -2,0,2 => 0
↦ -2,1,0 => 0
↦ -2,1,1 => 0
↦ -2,1,2 => 0
↦ -2,2,0 => 0
↦ -2,2,1 => 0
↦ -2,2,2 => 0
↦ -2,3,0 => 0
↦ -2,3,1 => 0
↦ -2,3,2 => 0
↦ -1,-2,0 => 0
↦ -1,-2,1 => 0
↦ -1,-2,2 => 0
↦ -1,-1,0 => 1
↦ -1,-1,1 => 0
↦ -1,-1,2 => 0
↦ -1,0,0 => 2
↦ -1,0,1 => 0
↦ -1,0,2 => 0
↦ -1,1,0 => 3
↦ -1,1,1 => 0
↦ -1,1,2 => 0
↦ -1,2,0 => 4
↦ -1,2,1 => 0
↦ -1,2,2 => 0
↦ -1,3,0 => 5
↦ -1,3,1 => 0
↦ -1,3,2 => 0
↦ 0,-2,0 => 0
↦ 0,-2,1 => 0
↦ 0,-2,2 => 0
↦ 0,-1,0 => 2
↦ 0,-1,1 => 0
↦ 0,-1,2 => 0
↦ 0,0,0 => 4
↦ 0,0,1 => 0
↦ 0,0,2 => 0
↦ 0,1,0 => 6
↦ 0,1,1 => 0
↦ 0,1,2 => 0
↦ 0,2,0 => 8
↦ 0,2,1 => 0
↦ 0,2,2 => 0
↦ 0,3,0 => 10
↦ 0,3,1 => 0
↦ 0,3,2 => 0
↦ 1,-2,0 => 0
↦ 1,-2,1 => 0
↦ 1,-2,2 => 0
↦ 1,-1,0 => 3
↦ 1,-1,1 => 0
↦ 1,-1,2 => 0
↦ 1,0,0 => 6
↦ 1,0,1 => 0
↦ 1,0,2 => 0
↦ 1,1,0 => 9
↦ 1,1,1 => 0
↦ 1,1,2 => 0
↦ 1,2,0 => 12
↦ 1,2,1 => 0
↦ 1,2,2 => 0
↦ 1,3,0 => 15
↦ 1,3,1 => 0
↦ 1,3,2 => 0
↦ 2,-2,0 => 0
↦ 2,-2,1 => 0
↦ 2,-2,2 => 0
↦ 2,-1,0 => 4
↦ 2,-1,1 => 0
↦ 2,-1,2 => 0
↦ 2,0,0 => 8
↦ 2,0,1 => 0
↦ 2,0,2 => 0
↦ 2,1,0 => 12
↦ 2,1,1 => 0
↦ 2,1,2 => 0
↦ 2,2,0 => 16
↦ 2,2,1 => 0
↦ 2,2,2 => 0
↦ 2,3,0 => 20
↦ 2,3,1 => 0
↦ 3,-2,0 => 0
↦ 3,-2,1 => 0
↦ 3,-2,2 => 0
↦ 3,-1,0 => 5
↦ 3,-1,1 => 0
↦ 3,-1,2 => 0
↦ 3,0,0 => 10
↦ 3,0,1 => 0
↦ 3,0,2 => 0
↦ 3,1,0 => 15
↦ 3,1,1 => 0
↦ 3,1,2 => 0
↦ 3,2,0 => 20
↦ 3,2,1 => 0
↦ 3,3,0 => 25
intvec d = 3,3,0;
cohomologytable*d;
↦ 25
def (Z,eulerTable) = eulerPolynomialTable(M,low,high);
setring(Z);
print(eulerTable);
↦ -3,-3 => h2
↦ -2,-3 => 0
↦ -1,-3 => h
↦ 0,-3 => 2h
↦ 1,-3 => 3h
↦ 2,-3 => 4h
↦ 3,-3 => 5h
↦ -3,-2 => 0
↦ -3,-1 => h
↦ -3,0 => 2h
↦ -3,1 => 3h
↦ -3,2 => 4h
↦ -3,3 => 5h
↦ -2,-2 => 0
↦ -2,-1 => 0
↦ -2,0 => 0
↦ -2,1 => 0
↦ -2,2 => 0
↦ -2,3 => 0
↦ -1,-2 => 0
↦ -1,-1 => 1
↦ -1,0 => 2
↦ -1,1 => 3
↦ -1,2 => 4
↦ -1,3 => 5
↦ 0,-2 => 0
↦ 0,-1 => 2
↦ 0,0 => 4
↦ 0,1 => 6
↦ 0,2 => 8
↦ 0,3 => 10
↦ 1,-2 => 0
↦ 1,-1 => 3
↦ 1,0 => 6
↦ 1,1 => 9
↦ 1,2 => 12
↦ 1,3 => 15
↦ 2,-2 => 0
↦ 2,-1 => 4
↦ 2,0 => 8
↦ 2,1 => 12
↦ 2,2 => 16
↦ 2,3 => 20
↦ 3,-2 => 0
↦ 3,-1 => 5
↦ 3,0 => 10
↦ 3,1 => 15
↦ 3,2 => 20
↦ 3,3 => 25
```

D.15.22.11 twist

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: twist(M,c); M module, c intvec

Purpose: twists the module M by c

Assume: M is a multigraded module

Return: M with the new grading

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec c = 1,1;
def (S,E) = productOfProjectiveSpaces(c);
setring(S);
module M = freemodule(2);
intmat gradeM[2][2] = 0,1,0,1;
M = setModuleGrading(M,gradeM);
getModuleGrading(M);
↦ 0,1,
↦ 0,1
intvec c = 2,2;
↦ // ** redefining c (intvec c = 2,2;) ./examples/twist.sing:9
module Mtwist = twist(M,c);
getModuleGrading(Mtwist);
↦ -2,-1,
↦ -2,-1
```

D.15.22.12 beilinsonWindow

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: beilinsonWindow(T); T multigradedcomplex

Purpsose: compute the subquotient complex of T consisting of summands generated in degrees 0 <= a <= n

Assume: T is a multigraded complex of free E-modules

Return: multigradedcomplex, the Beilinson window of T

Note: The returend summands are the only ones that contribute to the Beilinson monad.

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec f = 1,1;
def (S,E) = productOfProjectiveSpaces(f);
intvec low = -3,-3;
intvec high = 3,3;
setring(S);
module M = 0;
intmat MGrading[2][1] = -1,-1;
M = setModuleGrading(M,MGrading);
multigradedcomplex tate;
(E,tate) = tateResolution(M,low,high);
setring(E);
multigradedcomplex W = beilinsonWindow(tate);
W;
↦ 0  <--  E^4  <--  E^4  <--  E^1  <--  0
↦ -1      0         1         2         3
```

```
↦
intvec c = 1,1,1;
intvec low2 = 0,0,0;
intvec high2 = 0,1,0;
def (S2,E2) = productOfProjectiveSpaces(c);
setring(S2);
module M2 = 0;
intmat gradeM[3][1] = -1,-1,-1;
M2 = setModuleGrading(M2,gradeM);
multigradedcomplex tate2;
(E2,tate2) = tateResolution(M2,low2,high2);
setring(E2);
multigradedcomplex W2 = beilinsonWindow(tate2);
W2;
↦ 0  <--  E^8
↦ -1      0
↦
```

D.15.22.13 regionComplex

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: regionComplex(T,d,I,J,K); T multigradedcomplex, d intvec, I intvec, J intvec, K intvec

Purpose: compute the region complex of T w.r.t. the sets I,J,K and the vector d

Assume: I,J,K are intvecs representing disjoint subsets of {1,...,t}, T is a complex in ring E, zero represents the empty set

Return: multigraded complex which is the region complex T_d(I,J,K) of T

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec f = 1,1;
def (S,E) = productOfProjectiveSpaces(f);
intvec low = -3,-3;
intvec high = 3,3;
setring(S);
module M = 0;
intmat MGrading[2][1] = -1,-1;
M = setModuleGrading(M,MGrading);
multigradedcomplex tate;
(E,tate) = tateResolution(M,low,high);
setring(E);
tate;
↦ E^25  <--  E^40  <--  E^46  <--  E^44  <--  E^35  <--  E^30  <--  E^38  <\
  --  E^56  <--  E^81  <--  E^110  <--  E^141  <--  E^174  <--  E^210
↦ -6         -5         -4         -3         -2         -1         0      \
      1          2          3           4           5           6
↦
ring Z = cohomologyMatrixFromResolution(tate,low,high);
setring(Z);
print(cohomologymat);
↦ 5h,0,5,10,15,20,25,
↦ 4h,0,4,8, 12,16,20,
↦ 3h,0,3,6, 9, 12,15,
↦ 2h,0,2,4, 6, 8, 10,
↦ h, 0,1,2, 3, 4, 5,
↦ 0, 0,0,0, 0, 0, 0,
↦ h2,0,h,2h,3h,4h,5h
setring(E);
intvec  c= 0,-3;
intvec I = 0;
intvec J = 0,1;
intvec K = 0,2;
multigradedcomplex U = regionComplex(tate,c,I,J,K);
U;
↦ 0  <--  E^10  <--  E^8  <--  E^6  <--  E^4  <--  E^2  <--  E^2  <--  0
↦ -4      -3         -2        -1        0         1         2         3
↦
Z = cohomologyMatrixFromResolution(U,low,high);
setring(Z);
print(cohomologymat);
↦ 0,0,0,10,0,0,0,
↦ 0,0,0,8, 0,0,0,
↦ 0,0,0,6, 0,0,0,
↦ 0,0,0,4, 0,0,0,
↦ 0,0,0,2, 0,0,0,
↦ 0,0,0,0, 0,0,0,
↦ 0,0,0,2h,0,0,0
setring(E);
multigradedcomplex V = regionComplex(tate,c,I,J,J);
↦    ? I,J,K have to be disjoint.
↦    ? leaving tateProdCplxNegGrad.lib::regionComplex (0)
```

D.15.22.14 strand

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: strand(T,c,J)

Purpose: compute the strand of T w.r.t. the set J and the vector c

Return: subquotient complex of T which is the strand of T

Example:

```
LIB "tateProdCplxNegGrad.lib";
intvec f = 1,1;
def (S,E) = productOfProjectiveSpaces(f);
intvec low = -3,-3;
intvec high = 3,3;
setring(S);
module M = 0;
intmat MGrading[2][1] = -1,-1;
M = setModuleGrading(M,MGrading);
multigradedcomplex tate;
(E,tate) = tateResolution(M,low,high);
setring(E);
```

```
ring Z = cohomologyMatrixFromResolution(tate,low,high);
setring(Z);
print(cohomologymat);
↦ 5h,0,5,10,15,20,25,
↦ 4h,0,4,8, 12,16,20,
↦ 3h,0,3,6, 9, 12,15,
↦ 2h,0,2,4, 6, 8, 10,
↦ h, 0,1,2, 3, 4, 5,
↦ 0, 0,0,0, 0, 0, 0,
↦ h2,0,h,2h,3h,4h,5h
setring(E);
intvec  c= 0,-3;
intvec J = 1;
multigradedcomplex U = strand(tate,c,J);
U;
↦ 0  <--  E^10  <--  E^8  <--  E^6  <--  E^4  <--  E^2  <--  E^2  <--  E^4 \
   <--  E^6  <--  E^8  <--  E^10
↦ -4      -3         -2        -1        0         1         2         3   \
        4         5         6
↦
Z = cohomologyMatrixFromResolution(U,low,high);
setring(Z);
print(cohomologymat);
↦ 0,0,0,10,0,0,0,
↦ 0,0,0,8, 0,0,0,
↦ 0,0,0,6, 0,0,0,
↦ 0,0,0,4, 0,0,0,
↦ 0,0,0,2, 0,0,0,
↦ 0,0,0,0, 0,0,0,
↦ 0,0,0,2h,0,0,0
```

D.15.22.15 firstQuadrantComplex

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: firstQuadrantComplex(T,c); T multigradedcomplex, c intvec

Purpose: compute the first quadrant complex of T w.r.t. the set J and the vector c

Return: subquotient complex of T which is the first quadrand complex of T

D.15.22.16 lastQuadrantComplex

Procedure from library `tateProdCplxNegGrad.lib` (see Section D.15.22 [tateProdCplxNegGrad_lib], page 2551).

Usage: lastQuadrantComplex(T,c); T multigradedcomplex, c intvec

Purpose: compute the last quadrant complex of T w.r.t. the set J and the vector c

Return: subquotient complex of T which is the strand of T

D.15.23 VecField_lib

Library: VecField.lib

Purpose: vector fields, with algorithms for jordan and diagonal forms

Authors: Adrian Rettich, rettich@mathematik.uni-kl.de
Raul Epure, epure@mathematik.uni-kl.de

References:
[1] Kyoji Saito, Quasihomogene isolierte
Singularitaeten von Hyperflaechen, 1971

Overview: Implements a class VecField, represented by a vector. For example, 'VecField V = [x3,xy]' declares the vector field v = x3 d_x+xy d_y. Instead of a vector, an nx1 matrix is also accepted. The vector can be recovered as V.vec.
Supports coordinate transformations (via maps), which are represented by tracking a map 'V.coordinates' which maps the standard coordinates to those in which V is currently represented. V.dimension stores the vector field's dimension, which is just nvars(basering), and V.lin yields the linear part of V. You may set an additional parameter V.precision, which dictates the degree to which operations on the vector field should be exact.
The default precision is 1. Precision is preserved across transformations, additions, and all other manipulations of vector fields.

Procedures:

D.15.23.1 applyVecField

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: applyVecField(VecField V, poly/ideal p [,int n])

Return: if p is a polynomial, return a polynomial V(p), if p is an ideal, the ideal V(p). If an optional n is given, only the parts of V up to and including degree n are applied to p.

Example:

```
LIB "VecField.lib";
int oldp = printlevel;
printlevel = 1;
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
poly f = x3+2y3+xz2;
poly g = V*f;
ideal I = (x2,y);
ideal J = V*I;
printlevel = oldp;
```

D.15.23.2 changeCoordinates

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: changeCoordinates(vecField V, map psi), where psi is an algebra morphism k[x1,...,xn]->k[y1,...,ym] expressing the new basis in terms of the old.

Return: A new vecfield in the new coordinates.

Note: the coordinate change necessitates inverting psi.
The inversion will be correct up to the precision of V.

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
```

D.15.23.3 jordanVecField

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: jordanVecField(VecField V)

Return: new vecfield W in coordinates s.t. W.lin is in Jordan normal form.

Assume: eigenvalues of V.lin in basefield.

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
VecField X = jordanVecField(W);
```

D.15.23.4 diagonalizeVecFieldLin

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: diagonalizeVecFieldLin(list l), where l is a list of VecFields

Return: list W of the same VecFields in new coords s.t. the linear parts are diagonal, all in the same coordinates

Assume: All linear parts of the entries of l are simultaneously diagonalizable.

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
VecField X = [-2,1,0];
list d = diagonalizeVecFieldLin(W,X);
```

D.15.23.5 SaitoBase

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: SaitoBase(VecField V)

Return: new VecField W in the base from [1] thm. 3.1, where semisimple and nilpotent components are easily read off.

Note: the algorithm requires inversions of algebra morphisms. These will be exact to the precision of V. Warning: The algorithm assumes standard coordinates. If V is not given in standard coordinates, it will be converted to standard coordinates, but not converted back at the end, so the resulting transformation is from standard coordinates to the Saito coordinates. If you want the entire transformation from your original coordinates to the Saito ones, add the inverse of your original coordinate transformation manually.
Note that weight vectors only take Int64,
making the algorithm fail for very large entries in V.vec.

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
VecField WS = SaitoBase(W);
```

D.15.23.6 diagonalizeVecField

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: diagonalizeVecField(list l), l a list of VecFields

Return: list of VecFields W such that they are all simultaneously diagonal.

Assume: all input VecFields have to be simultaneously diagonalizable.

Note: the algorithm requires inversions of algebra morphisms. These will be exact to the precision of l[1]. Warning: The algorithm assumes standard coordinates. If V is not given in standard coordinates, it will be converted to standard coordinates, but not converted back at the end, so the resulting transformation is from standard coordinates to the Saito coordinates. If you want the entire transformation from your original coordinates to the Saito ones, add the inverse of your original coordinate transformation manually.
Note that weight vectors only take Int64,
making the algorithm fail for very large entries in V.vec.

Example:

```
LIB "VecField.lib";
int oldp = printlevel;
printlevel = 1;
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
VecField X = [-2,1,0];
VecField WS = diagonalizeVecField(W,X);
↦    ? error in diagonalizeVecField: no solutions found
↦    ? leaving VecField.lib::diagonalizeVecField (0)
printlevel = oldp;
```

D.15.23.7 vecFieldToMatrix

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: vecFieldToMatrix(VecField V,ideal W)

Return: the matrix representation of V restricted to span(W), in the basis specified by W.

Example:

```
LIB "VecField.lib";
int oldp = printlevel;
printlevel = 1;
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
matrix m = vecFieldToMatrix(W,maxideal(2));
printlevel = oldp;
```

D.15.23.8 decomposeVecField

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: decomposeVecField(VecField V) or decomposeVecField(list s), s a list of VecFields

Return: list l containing two VecFields, where l[1] is the semisimple part of V, l[2] the nilpotent part. If called with a list, l[2n-1] is the semisimple part of the n-th VecField, l[2n] its nilpotent part.

Assume: The algorithm assumes standard coordinates. If V is not given in standard coordinates, it will be converted to standard coordinates, but not converted back at the end, so the resulting transformation is from standard coordinates to the new coordinates. If you want the entire transformation from your original coordinates to the new ones, add the inverse of your original coordinate transformation manually.

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
VecField W = changeCoordinates(V,phi);
list l = decomposeVecField(W);
```

D.15.23.9 diagonalizeMatrixSimul

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: diagonalizeMatrixSimul(list l), where l is a list of quadratic matrices of same dimensions

Return: trafo matrix U s.t. U*A*inv(U) is diagonal for each A in l

Assume: matrices are simultaneously diagonalizable, nvars(basering)>=dimension of the matrices

Example:

```
LIB "VecField.lib";
ring r = 0,(x(1),x(2),x(3),x(4)),dp;
matrix a[4][4] = 2,0,0,0, 0,3,0,0, 0,0,3,0, 0,0,0,5 ;
matrix b[4][4] = 6,0,0,0, 0,6,0,0, 0,0,2,0, 0,0,0,7 ;
matrix trafo[4][4] = 3,3,0,0, 0,1,3,0, 2,0,1,0, 0,2,2,3;
matrix invtrafo = inverse(trafo);
matrix at = invtrafo * a * trafo;
matrix bt = invtrafo * b * trafo;
list l;
l = insert(l,at);
l = insert(l,bt);
matrix diagtrafo = diagonalizeMatrixSimul(l);
```

D.15.23.10 invertAlgebraMorphism

Procedure from library `VecField.lib` (see Section D.15.23 [VecField_lib], page 2571).

Usage: invertAlgebraMorphism(map p,int n), where p is an algebra morphism k[x1,...,xn]->k[y1,...,ym],
or invertAlgebraMorphism(ideal p,int n) if you represent the map as an ideal

Return: the inverse of p mod (maxideal)^n.
If n=-1, try for infinite precision. If input was an ideal, return an ideal.

Note: the algorithm used is described in [1], chapter 3.2

Example:

```
LIB "VecField.lib";
ring r = 0, (x, y, z),ds;
vector v = [-1,-1,-1];
VecField V = v;
V.precision = 4;
map phi = r, x-2y2+z3,2y+y3+z,z;
map phiinv = invertAlgebraMorphism(phi,4);
```

8 Release Notes

8.1 News and changes

NEWS in SINGULAR 4.3.2

News for version 4.3.2

New libraries:

`sagbiNormaliz.lib`: computation of Sagbi bases via normaliz (Section D.15.18 [sagbiNormaliz_lib], page 2541)

`stdmodule.lib`: Compute Standard Bases of submodule of free module over polynomial subalgebra (⟨undefined⟩ [stdmodule_lib], page ⟨undefined⟩)

Changed libraries:

`normal.lib`: optimized, integrated `locnorma.lib` and `modnormal.lib` (Section D.4.23 [normal_lib], page 1183)

`normaliz.lib`: update to normaliz 3.10 (Section D.4.24 [normaliz_lib], page 1209)

`hess.lib`: fixes (Section D.5.7 [hess_lib], page 1448)

`sheafcoh.lib`: new routines Section D.5.18.5 [sheafCohBGGregul], page 1609, Section D.5.18.6 [sheafCohBGGregul_w], page 1610, Section D.5.18.9 [sheafCoh_w], page 1617, (Section D.5.18 [sheafcoh_lib], page 1604)

`sheafcoh.lib`: display of the tables is moved to Section D.5.18.12 [displayCohom], page 1619 (Section D.5.18 [sheafcoh_lib], page 1604)

`elim.lib`: rename `sat` to `sat_with_exp`, `sat` now return the saturation i9deal (only).

`primdec.lib`: fixed a bug in absolute primary decomposition (Section D.4.26 [primdec_lib], page 1239)

New type:

`bigintvec`: Section 4.4 [bigintvec], page 78

New commands:

new command Section 5.1.123 [prune_map], page 245

new command Section 5.1.99 [mres_map], page 226

New/renamed monomial orderings:

To avoid confusion (with the degree reverse lexicographic ordering) the orderings `rp` and `rs` are renamed:

renamed orderings: `rp -> ip`, `rs -> is`

new ordering `Ip`

Changes in the kernel/build system:

new algorithm for Section 5.1.151 [std], page 271 in the case of ordering `ds` and zero dimensional ideal over `QQ`

new algorithm for lift function (Section 5.1.80 [lift], page 211) mstd for local rings (Section 5.1.100 [mstd], page 227)

use the new Hilbert function algorithm for Section 5.1.56 [hilb], page 195. (Different output format)

overflow in `hilb` is an error (on request of a single user)

renamed orderings: `ringorder_rp -> ringorder_ip`, `ringorder_rs -> ringorder_is`

new ordering `ringorder_Ip`

changes for FLINT 3.0.x

News for version 4-3-2

Changed libraries:

many: call std in many places only if attribute `isSB` is not set

many: change many calls to `execute` by better variants

New commands:

`coeffs` for `ring` (Section 5.1.12 [coeffs], page 165)

Changes in the kernel/build system:

new algorithm for Hilbert function (Section 5.1.56 [hilb], page 195)

News for version 4-3-1

New libraries:

`normal.lib`: new command `isNormal`, option `normalCheck` for `normal` (Section D.4.23 [normal_lib], page 1183)

Changed libraries:

chern.lib: fixed `symNsym` (Section D.5.2 [chern_lib], page 1390)

homolog.lib: fixed corner cases for `depth` (Section D.4.11 [homolog_lib], page 1090)

New commands:

`chinrem`, `farey` for `smatrix` (Section 4.21 [smatrix], page 129)

Changed commands:

overflow check for Section 5.1.168 [vdim], page 286

better overflow check/les overflows for Hilbert function (Section 5.1.56 [hilb], page 195)

`delete` accepts `intvec` for the indices to delete (Section 5.1.21 [delete], page 171)

several GB based commands accept the choice of the algorithm: `"std","slimgb","sba","modstd","singmatic","groebner","ffmod","nfmod"` (Section 5.1.156 [syz], page 280, Section 5.1.28 [eliminate], page 176, Section 5.1.65 [intersect], page 202, Section 5.1.94 [modulo], page 223, Section 5.1.81 [liftstd], page 212, Section 5.1.80 [lift], page 211)

`reduce` for non-field coefficients: search best reduction

Changes in the kernel/build system:

changes for gcc 12

new option `--log LOGFILE`

configure can change the path to hml documentation (`--htmldir=...`)

different format for `doc.tbz2`

new algorithm for Hilbert function (Section 5.1.56 [hilb], page 195)

News for version 4-3-0

New libraries:

enumpoints.lib: enumerating rational points (Section D.15.3 [enumpoints_lib], page 2328)

sagbigrob.lib: Sagbi-Groebner basis of an ideal of a subalgebra (Section D.15.17 [sagbigrob_lib], page 2538)

puiseuxexpansion.lib: Puiseux expansions over algebraic extensions (Section D.15.14 [puiseuxexpansions_lib], page 2532)

integralbasis_lib: Integral basis in algebraic function fields: new version (Section D.4.12 [integralbasis_lib], page 1114)

Changes in the kernel/build system:

input history is stored by default in `.singularhistory` (Section 3.1.5 [Editing input], page 18).

ABI change: all number routines (`n_...`) have only `coeffs` as last argument, functions with `ring` as last argument are removed

PATH is not changed for `system("sh",..)` (use ⟨undefined⟩ [system, SingularBin], page ⟨undefined⟩)

`hilb` avoids int overflow (also in `degree, stdhilb`)

`liftstd` (with 2 arguments) improved (Section 5.1.81 [liftstd], page 212)

`noether` improved (Section 5.3.5 [noether], page 304), use in `groebner(I,"HC")` for faster results for local orderings, 0-dimensional ideals (Section 5.1.53 [groebner], page 191).

letterplace routines improved (Section 7.7 [LETTERPLACE], page 616)

info file is now `singular.info` instead of `singular.hlp`

update for using FLINT 2.8.x

News for version 4-2-1

New commands:

Letterplace: dim, rightStd for qrings (Section 7.7 [LETTERPLACE], page 616)

Letterplace: map, fetch, imap (Section 4.12 [map], page 106, Section 5.1.38 [fetch], page 182, Section 5.1.59 [imap], page 198)

New libraries:

decomp.lib: functional decomposition of polynomials (Section D.4.6 [decomp_lib], page 1051)

hodge.lib: algorithms for Hodge ideals (Section D.15.7 [hodge_lib], page 2421)

tateProdCplxNegGrad.lib: sheaf cohomology on product of projective spaces (Section D.15.22 [tateProdCplxNegGrad_lib], page 2551)

Changes in the kernel/build system:

`liftstd` (with 2 arguments) improved (Section 5.1.81 [liftstd], page 212)

building on Cygwin with shared libraries

building the manual via `--enable-doc-build`

News for version 4-2-0

Syntax changes:

renamed poly.lib to polylib.lib (Section D.2.8 [polylib_lib], page 887)

New libraries:

interval.lib: interval arithmetic (Section D.8.2 [interval_lib], page 1824)

maxlike.lib: algebraic statistics (Section D.15.9 [maxlike_lib], page 2434)

nchilbert.lib: Hilbert series for LetterPlace algebras (Section 7.5.13 [nchilbert_lib], page 517)

polyclass.lib: class of polynomials (Section D.15.13 [polyclass_lib], page 2529)

recover.lib: Hybrid numerical/symbolical algorithms (Section D.8.7 [recover_lib], page 1861)

redcgs.lib: Reduced Comprehensive Groebner Systems (Section D.2.9 [redcgs_lib], page 897)

ringgb.lib: coefficient rings (Section D.15.15 [ringgb_lib], page 2535)

sets.lib: Sets (Section D.14.8 [sets_lib], page 2268)

stanleyreisner.lib: T1 and T2 for a general Stanley-Reiser ring (Section D.15.19 [stanleyreisner_lib], page 2543

systhreads.lib: multi-threaded objects (Section D.15.21 [systhreads_lib], page 2551)

Changed libraries:

classify_aeq.lib: new procedure `classSpaceCurve` (Section D.6.6 [classify_aeq_lib], page 1656)

grobcov.lib: new version (Section D.2.4 [grobcov_lib], page 817)

modular.lib: parallel version for verification via `system("verifyGB",I)`

New commands:

`system("verifyGB",I)`: test, if I is a Groebner basis (using parallel processes)

Letterplace: modulo,syz,lift,liftstd, rightStd (Section 7.7 [LETTERPLACE], page 616)

Changes in the kernel/build system:

update for using FLINT 2.6.x and for FLINT 2.7.0

Singular can be build with NTL or FLINT or both (if none is available, `factorize` and `gcd` will not work.)

News for version 4-1-3

New libraries:

invar.lib: Invariant theory Section D.7.4 [invar_lib], page 1812

moddiq.lib: ideal quotient and saturation Section D.4.14 [moddiq_lib], page 1120

ncModslimgb.lib: modular Groebner bases for G-algebras Section 7.5.18 [ncModslimgb_lib], page 553

Changed libraries:

chern.lib: new version (Section D.5.2 [chern_lib], page 1390)

grobcov.lib: new version (Section D.2.4 [grobcov_lib], page 817), new functions Section D.2.4.22 [ConsLevels], page 869, Section D.2.4.23 [Levels], page 870, Section D.2.4.24 [Grob1Levels], page 872, Section D.2.4.25 [DifConsLCSets], page 875

Changes in the kernel/build system:

improved gcd and multiplication via FLINT improved lift (and related)

port to polymake 3.5.x

rational functions via flint (Section 5.1.44 [flintQ], page 186)

free algebra over Z (Section 7.7 [LETTERPLACE], page 616)

adaptions/functions for `Singular.jl`(https://github.com/oscar-system/Singular.jl)

News for version 4-1-2

New libraries:

arnoldclassify.lib: Arnol'd Classifier of Singularities (Section D.6.3 [arnoldclassify_lib], page 1639)

difform.lib: Procedures for differential forms (Section D.15.2 [difform_lib], page 2291)

dmodideal.lib: Algorithms for Bernstein-Sato ideals of morphisms (Section 7.5.6 [dmodideal_lib], page 446)

fpalgebras.lib: Generation of various algebras in the letterplace case (Section 7.10.2 [fpalgebras_lib], page 645)

ncrat.lib: non-commutatie rational functions (Section 7.10.6 [ncrat_lib], page 675)

Changed libraries:

freegb.lib: lpDivision, lpPrint (Section 7.10.4 [freegb_lib], page 665)

fpadim.lib (Section 7.10.1 [fpadim_lib], page 639)

schreyer.lib: deprecated

goettsche.lib: new, extended version (The Nakajima-Yoshioka formula up to n-th degree,Poincare Polynomial of the punctual Quot-scheme of rank r on n planar points Betti numbers of the punctual Quot-scheme of rank r on n planar points)(Section D.5.5 [goettsche_lib], page 1437)

grobcov.lib: small bug fix (Section D.2.4 [grobcov_lib], page 817)

Changes in the kernel/build system:

integrated xalloc into omalloc: (`./configure --disable-omalloc`)

improved heuristic for `det` (Section 5.1.23 [det], page 172)

improved reading of long polynomials

improved groebner bases over Z coefficients

code for free algebras (letterplace rings) rewritten (using now the standrad `+,-,*,^,std`,...) (Section 7.7 [LETTERPLACE], page 616)

new commands `rightstd` (Section 7.8.10 [rightstd (letterplace)], page 632)

extended `twostd` to Letterplace (Section 7.8.14 [twostd (letterplace)], page 634, Section 7.3.29 [twostd (plural)], page 363)

pseudo type `polyBucket`

new type `smatrix`: sparse matrix (experimental) (Section 4.21 [smatrix], page 129).

extended `coef` to ideals (Section 5.1.11 [coef], page 164).

error and signal handling in `libSingular` (Section 8.3 [libSingular], page 2588).

updated gfanlib to version 0.6.2

port to NTL 11 (needs C++11: gcc6 or -std=c++11), which does not conflict with polymake (needs C++14)

News for version 4-1-1

New syntax:

`alias`: may be used as a prefix to a variable declaration. Can only be used in procedure headings. (Section 3.5.1 [General command syntax], page 41).

New command:

`fres`: improved version of `sres`: computes a (not necessarily minimal) free resolution of the input ideal/module, using Schreyer's algorithm. (Section 5.1.48 [fres], page 188,Section 5.1.149 [sres], page 269).

Extended commands:

pseudo ordering `L` allows setting of limits for exponents in polynomials (Section B.2.9 [Pseudo ordering L], page 773,Section 5.1.2 [attrib], page 156 for `maxExp`)

`%`,`mod`: also for poly operands (Section 4.17.3 [poly operations], page 121).

`delete`: extended to intvec, ideal, module (Section 5.1.21 [delete], page 171).

syz (Section 5.1.156 [syz], page 280), lift (Section 5.1.80 [lift], page 211), liftstd (Section 5.1.81 [liftstd], page 212), intersect(Section 5.1.65 [intersect], page 202): with a specified GB algorithm

New libraries:

classify2.lib: Classification of isolated singularities of corank <=2 and modality <= wrt. right equivalence over the complex numbers according to Arnold's list. (Section D.6.5 [classify2_lib], page 1654)

goettsche.lib: Goettsche's formula for the Betti numbers of the Hilbert scheme of points on a surface, Macdonald's formula for the symmetric product (Section D.5.5 [goettsche_lib], page 1437)

combinat.lib, modules.lib, methods,lib, nets.lib: a more mathematical view of modules (Section D.14.2 [combinat_lib], page 2232: combinatorics), (Section D.14.4 [methods_lib], page 2239: construct procedures), (Section D.4.15 [modules_lib], page 1122: free resolutions), (Section D.14.5 [nets_lib], page 2239: pretty printing)

ncHilb.lib: Hilbert series of non-commutative monomial algebras (Section 7.10.5 [ncHilb_lib], page 672)

realclassify.lib: Classification of real singularities(Section D.6.19 [realclassify_lib], page 1745)

rootisolation.lib: real root isolation using interval arithmetic(Section D.8.8 [rootisolation_lib], page 1889)

rstandard.lib: Janet bases and border bases for ideals (Section D.4.31 [rstandard_lib], page 1265)

Changed libraries:

chern.lib: new version (Section D.5.2 [chern_lib], page 1390)

gitfan.lib: new (incompatible) version (Section D.13.3 [gitfan_lib], page 2089)

grobcov.lib: new version (Section D.2.4 [grobcov_lib], page 817)

Changes in the kernel/build system:

port to polymake 3.x.x

port to NTL 10 with threads (needs also C++11: gcc6 or -std=c++11)

p_Invers is only a helper for p_Series: now static

p_Divide is now p_MDivide, pDivide/p_Divide is a new routine

News for version 4-1-0

Syntax changes:

new (additional) form of ring definitions: (for example `ring R=QQ[x,y,z];`) (Section 3.3.2 [General syntax of a ring declaration], page 32)

new (additional) form of multi-indicies: (for example `i(1,2,3,4,5)`) (Section 3.5.3 [Names], page 44)

changed behaviour of `charstr` (Section 5.1.7 [charstr], page 162)

new data type `cring` to describe the coefficient rings, to be used for the new definitions for (polynomial) rings (Section 3.3.2 [General syntax of a ring declaration], page 32)

new command `ring_list` to access the parts used to construct polynomial rings (Section 5.1.138 [ring_list], page 257,Section 5.1.137 [ringlist], page 255)

extended polynomial ring construction: also from lists produced by `ring_list`

new attribute `ring_cf` for `ring` (Section 5.1.2 [attrib], page 156)

printing of rings changed to match `cring` names (Section 5.1.7 [charstr], page 162)

New libraries:

new library: classifyMapGerms.lib: standard basis of the tangent space at the orbit of an algebraic group action (Section D.6.9 [classifyMapGerms_lib], page 1662)

new library: ffmodstd.lib: Groebner bases of ideals in polynomial rings over algebraic function fields(Section D.4.9 [ffmodstd_lib], page 1080)

new library: nfmodsyz.lib: syzygy modules of submodules of free modules over algebraic number fields(Section D.4.21 [nfmodsyz_lib], page 1177)

new library: curveInv.lib: invariants of curves (Section D.4.5 [curveInv_lib], page 1048)

new library: gfan.lib: interface to gfanlib (Section D.13.2 [gfan_lib], page 2049)

extended library: interface to polymake merged into Section D.13.4 [polymake_lib], page 2128

new library: tropicalNewton.lib: Newton polygon methods in tropical geometry (Section D.13.7 [tropicalNewton_lib], page 2195)

new library: schubert.lib: some procedures for intersction theory (Section D.5.17 [schubert_lib], page 1586)

Changed libraries:

classify_aeq.lib: new procedures (Section D.6.6 [classify_aeq_lib], page 1656)

grobcov.lib: new version (Section D.2.4 [grobcov_lib], page 817)

ncfactor.lib: factorization in some noncommuative algebras (Section 7.5.12 [ncfactor_lib], page 486) with new routine ncfactor (Section 7.5.12.1 [ncfactor], page 486)

primdec.lib: new option "subsystem" (Section D.4.26 [primdec_lib], page 1239)

Changes in the kernel:

improved mapping of polynomials/ideals/...

port to gcc 6

port to gfanlib 0.6 (requires C++11, i.e. gcc >=4.3)

port to NTL 10

port to polymake 3.0

port to readline 7

Section 5.1.140 [sba], page 259 works for global orderings, also for coefficient types Z and Z/m

Section 5.1.151 [std], page 271 works for all orderings, also for coefficient types Z and Z/m with local/mixed orderings

Section 5.1.36 [factorize], page 180 works for polynomial rings over ZZ

Experimental stuff:

module Section D.14.3 [customstd_lib], page 2237: modify `std` (Section D.14.3.2 [satstd], page 2238)

News for version 4-0-3

New libraries:

new library: brillnoether.lib: Riemann-Roch spaces of divisors on curves (Section D.5.1 [brillnoether_lib], page 1389)

new library: chern.lib: Chern classes (Section D.5.2 [chern_lib], page 1390)

new library: ffmodstd.lib: Groebner bases of ideals in polynomial rings over algebraic function fields(Section D.4.9 [ffmodstd_lib], page 1080)

new library: GND.lib: General Neron Desingularization (Section D.15.5 [GND_lib], page 2342)

new library: graal.lib: localization at prime ideals (Section D.5.6 [graal_lib], page 1443)

new library: hess.lib: Riemann-Roch space of divisors (Section D.5.7 [hess_lib], page 1448)

Changed libraries:

renamed algemodstd_lib to Section D.4.20 [nfmodstd_lib], page 1174, extended to `module`

renamed derham_lib to Section D.5.3 [deRham_lib], page 1426

grobcov.lib (grobcovK): Groebner Cover for parametric ideals (Section D.2.4 [grobcov_lib], page 817) with new routine ConsLevels (Section D.2.4.22 [ConsLevels], page 869), removed AddCons AddConsP.

News for version 4-0-2

New commands:

align (Section 5.1.1 [align], page 156)

branchTo (Section 4.18.3 [procs with different argument types], page 125)

`->` (Section 4.18.2 [proc expression], page 125)

Change in ring handling:

`typeof(` qring `)` returns `"ring"`

New libraries:

algemodstd.lib: Groebner bases of ideals in polynomial rings over algebraic number fields(renamed to Section D.4.20 [nfmodstd_lib], page 1174)

arr.lib: arrangements of hyperplanes (Section D.14.1 [arr_lib], page 2205)

brillnoether.lib: Riemann-Roch spaces of divisors on curve (Section D.5.1 [brillnoether_lib], page 1389)

hess.lib: Riemann-Roch space of divisors on function fields and curves (Section D.5.7 [hess_lib], page 1448)

gradedModules.lib: graded modules/matrices/resolutions (Section D.15.6 [gradedModules_lib], page 2345)

Changed libraries:

revised polymake interface (polymake.so)

revised gfanlib interface (gfanlib.so)

Presolve::findvars (Section D.8.3.6 [findvars], page 1833, Section 5.1.166 [variables], page 285)

Ring::addvarsTo (Section D.2.12.26 [addvarsTo], page 956)

Ring::addNvarsTo (Section D.2.12.27 [addNvarsTo], page 957)

Ring::hasAlgExtensionCoefficient (Section D.2.12.21 [hasAlgExtensionCoefficient], page 955)

Schreyer::s_res (`s_res`)

grobcov.lib (grobcovK) (Section D.2.4 [grobcov_lib], page 817) with new routines AddCons AddConsP.

normaliz.lib (for normaliz >=2.8) (Section D.4.24 [normaliz_lib], page 1209)

renamed groebnerFan to groebnerFanP in polymake.lib (Section D.13.4 [polymake_lib], page 2128)

renamed fVector to fVectorP in polymake.lib (Section D.13.4 [polymake_lib], page 2128,`polymakeInterface_lib`)

News for version 4-0-1

Version 4-0-1 is a bug fix release.

New feature: attribute `ring_cf` for `ring` (Section 5.1.2 [attrib], page 156)

News for version 4-0-0

Version 4-0-0 is a milestone release of Singular. The new release series 4 aims for an entirely modularized architecture simplifying connectivity with other systems and paving the way for parallel computations. As a first step in modularization, the new release features an internal structural separation of coefficient rings and polynomial rings. This allows for a flexible integration of new coefficient rings.

Singular 4-0-0's list of new functionality and significant improvements further extends that of the 3-1-6/7 prerelease series.

New functionality

de Rham cohomology of complements of algebraic varieties (Section D.5.3 [deRham_lib], page 1426)

Gromov-Witten numbers of elliptic curves (Section D.4.8 [ellipticcovers_lib], page 1067)

classification of isolated complete intersection singularities in characteristic 0 (Section D.6.8 [classifyci_lib], page 1661)

parametrization of orbits of unipotent actions (Section D.5.10 [orbitparam_lib], page 1458)

F5-like Groebner basis algorithm (Section 5.1.140 [sba], page 259)

element-wise application of functions to data structures (Section 5.2.1 [apply], page 290)

support for debugging libraries (Section 3.9.1 [ASSUME], page 68)

Improved functionality

Groebner cover for parametric ideals (Section D.2.4 [grobcov_lib], page 817)

normalization of affine rings (Section D.4.23 [normal_lib], page 1183)

classification of real singularities (Section D.6.19 [realclassify_lib], page 1745)

GIT-fans (Section D.13.3 [gitfan_lib], page 2089) algebraic/transcendental field extensions

Chapter 7 [Non-commutative subsystem], page 317

an abstraction layer for parallel computations (Section D.2.7 [parallel_lib], page 884)

run-time loading of supplementary kernel code (Section A.1.9 [Dynamic modules], page 708)

interpreter language support for name spaces (Section 4.16 [package], page 119)

Availability

SINGULAR is available as source code and for Linux, Mac OS X, Windows, FreeBSD and SunOS-5.

8.2 Singular 3 and Singular 4

The purpose of this section is to describe new features and changes between Singular 3-1-7 and Singular 4.* (formerly known as Spielwiese) both for developers and Singular users. In what follows we will refer to the systems as Singular 3 and Singular 4.

8.2.1 Version schema for Singular

SINGULAR version is of the form `a.b.c.d` which may also be written as `a-b-c-d` where a,b,c and d are numbers:

a is changed with major, incompatible changes

b is changed with incompatible changes (of some commands/libraries)

c is changed with compatible changes (i.e. new commands, extended options, new algorithms, etc.)

d is changed with each release (i.e. with bug fixes, etc.)

SINGULAR does also have "unofficial" build originating from a code version between "official" version: such builds display "Development version a.b.c" in the header while "official" versions show "version a.b.c". Also the manual describes version a-b-c. To get the complete version number, use `system("version");` or use `SINGULAR_VERSION` in C.

8.2.2 Notes for Singular users

Coefficient rings

To allow for easy integration of new coefficient rings into Singular, the the way coefficient rings are being handled has been redesigned.

In general, the user syntax has not changed, however there are some changes in the behaviour of Singular:

- setting `minpoly` results in changing the current coefficient domain and clears all previously defined variables of that ring
- Minor changes in the output of coefficient ring description. Moreover the output of elements of certain rings has been improved (for example, reals).
- Algebraic and transcendental extensions of rationals and finite fields have been reimplemented. In particular, the heuristics for clearing denominators and factoring out content have been changed. In some cases this leads to a different, mathematically equivalent results of Groebner bases and related computations. For example a Groebner basis element may differ by a unit.
- Most notably, due to the redesign of the coefficient rings, if the user sets the minimal polynomial all variables dependent on the current ring are deleted.

Ring-dependent options

Formally global Singular Section 5.1.111 [option], page 234 now belong to individual polynomial rings. This includes:

- `intStrategy`
- `redTail`
- `redThrough`

Also the following settings now belong to individual (currently active) polynomial rings:

- `short`
- `minpoly`
- `noether`

Hence setting these options only affects the current ring. Be aware of this when switching between different rings, since the options affect the result of various computations (in particular Groebner bases).

Path names

- The tree structure of the binary Singular distribution has been changed. The typical tree now looks as show at `https://github.com/Singular/Singular/wiki/Sw-tree`
- Accordingly Singular search paths (where Singular searches for libraries, dynamic modules, etc.) have been changed. You can display them by calling Singular by `Singular -v`.
- currently, multi-arch installations of Singular 4 aere not possible.

Library versioning

Due to switching from Subversion to GIT revision control system for the Singular source code, library version variables (displayed when loading a library) have changed.

New orderings for modules

The now can assign weights to module components, when defining a monomial ordering. For example

```
ring R = 0, (x,y,z), (am(1,2,3,   10,20,30,40), dp, C);
deg(x*gen(1));
↦ 11
```

will assign weights 1,2,3 to x,y,z respectively, and weights 10,20,30,40,0,0,... to components of any free module defined over R. This ordering will first sort by this weighted degree, then by dp on the ring monomials and then will give priority to the large component index.

Future benefits of Singular 4

The redesign of Singular will allow us to provide new features in the future, for example:

- Interpreter type for coefficient rings.
- User defined coefficient rings.
- Improved syntax for defining polynomial rings.

8.2.3 Notes for developers

There has been an intensive process of refactoring, redesign and modularization of Singular to facilitate easier maintenance and future development:

- Build System : automake, libfac has been integrated into Factory
- Removed MP (Multi protocol) in favor of SSI links.
- Separation/modularization into libraries and packages
- For easy integration of new coefficient rings, we defined a generic interface for coefficient rings and a supporting framework for making them accessible to the user.

  In particular we have separated everything related to coefficient rings into a separate library `libcoeffs`. Dependency tree between restructured packages is show at `https://www.singular.uni-kl.de/dox/singular.png`

  In order to use `libSingular` as a C++ library, see Section 8.3 [libSingular], page 2588.

8.2.4 Building Singular

The user can build and install Singular with the following standard UNIX-like procedure:

- Download and extract the latest official source package (.tar.gz).
- Run the configure script, for instance, `./configure`.
- Build Singular by running `make`.
- Install Singular by running `make install`.

In contrast to Singular 3, there are now many more configuration options.

All possible options for configure can be seen by running the configure script with option `--help`. On a multicore compute consider running make with the option `-j [cores]`.

8.2.5 Side-by-side installation

Due to choosing paths according to FS standards it is no longer possible to have a side-by-side installation of different Singular versions or versions for different architectures.

8.3 libSingular

`libSingular` is the C++-library version of Singular.
`Singular/libsingular.h` is the main include file, `-lSingular` the link parameter,
`lib/pkgconfig/Singular.pc` provides all parameters in the pkconfig format.

It contains all parts of Singular with the following exceptions:

1. memory allocation functions for GMP (see `mmInit` in `Singular/tesths.cc`)
2. signal handlers (see `init_signals` in `Singular/cntrlc.cc`).
   At least a handler for `SIGCHLD` must be installed for the commands Section 5.1.170 [waitfirst], page 287, Section 5.1.169 [waitall], page 286 and the routines from Section D.2.7 [parallel_lib], page 884, Section D.4.16 [modstd_lib], page 1146, Section D.4.23 [normal_lib], page 1183, Section D.2.13 [tasks_lib], page 958.
   If the child was started by `libSingular` the handler has to call `sig_chld_hdl` from `Singular/links/ssiLink.cc` or implement something similar (call `slClose(l)` for ssi links).
3. error handlers for factory, NTL (see `init_signals` in `Singular/cntrlc.cc`).

8.4 Download instructions

Singular is available as source and binary program for most common hard- and software platforms. Instructions to download and install Singular can be found at

https://www.singular.uni-kl.de/index.php/singular-download.html.

Sources od release versions of Singular are also available from our FTP site

ftp://jim.mathematik.uni-kl.de/pub/Math/Singular/src/.

or

https://wwww.singular.uni-kl.de/ftp/pub/Math/Singular/src/.

8.5 Used environment variables

Singular needs to find some files (dynamic modules, libraries, help files). Usually they are found relative to the location of the (main) executable (after following symlinks). This can be changed by setting the following environment variables.

SINGULAR_EXECUTABLE (should usually not be set)
the complete filename of the main executable, usually derived from the command line (inspecting also PATH, following symlinks).
If SINGULAR_EXECUTABLE cannot be found, $prefix/bin/Singular is assumed.
For `libSingular`: SINGULAR_EXECUTABLE is set to the argument of `siInit` (it must exist).

SINGULAR_BIN_DIR
the directory of the main executable, usually derived from $SINGULAR_EXECUTABLE

SINGULAR_ROOT_DIR
the root of the singular tree, default: $SINGULAR_BIN_DIR/..

SINGULAR_DATA_DIR
the root of the singular data files, default: $SINGULAR_BIN_DIR/../share

SINGULARPATH
the directories for libraries and optional dynamic modules (separated by `;`), default:

$SINGULAR_DATA_DIR/singular/LIB

$SINGULAR_ROOT_DIR/share/singular/LIB

$SINGULAR_BIN_DIR/../share/singular/LIB

$SINGULAR_DATA_DIR/factory

$SINGULAR_ROOT_DIR/share/factory

$SINGULAR_BIN_DIR/LIB

$SINGULAR_BIN_DIR/../factory

$SINGULAR_BIN_DIR/MOD

$SINGULAR_ROOT_DIR/lib/singular/MOD

$SINGULAR_ROOT_DIR/libexec/singular/MOD

$prefix/lib/singular/MOD

$prefix/libexec/singular/MOD

$SINGULAR_BIN_DIR

SINGULAR_PROCS_DIR
the directories for dynamic modules (separated by `;`), default:

$SINGULAR_BIN_DIR/MOD

$SINGULAR_ROOT_DIR/lib/singular/MOD
$SINGULAR_ROOT_DIR/libexec/singular/MOD
$prefix/lib/singular/MOD
$prefix/libexec/singular/MOD

SINGULAR_INFO_FILE
`singular.info`, default: $SINGULAR_DATA_DIR/info/singular.info

SINGULAR_IDX_FILE
the help index, default: $SINGULAR_DATA_DIR/singular/singular.idx

SINGULAR_HTML_DIR
the directory of the manual as html files, default: $SINGULAR_DATA_DIR/singular/html

SINGULAR_URL
the URL of the manual, default: https://www.singular.uni-kl.de/Manual/

The effective list of directories/files can be printed by `Singular -v`, see Section 3.1.6 [Command line options], page 19.

Depending on the used functions, these environment variables apply also to `libSingular`.

8.6 Unix installation instructions

Install binaries: `https://www.singular.uni-kl.de/index.php/singular-download/install-linuxunix`
or build it yourself:

Install the necessary packages:

libtool
gnu make
gcc, g++
libreadline
gmp, mpfr
ntl
libcdd

Install flint 2.5 (or newer): `./configure --with-gmp=/usr --prefix=$HOME/tmp`
`make && make install`

Install Singular `./configure --with-flint=$HOME/tmp --enable-gfanlib --prefix=$HOME/Singular4`
`make && make install`
(`$prefix/bin/Singular` is the main executable)

(optional) install 4ti2

(optional) install surf/surfer

(optional) install normaliz 2.8 (or newer)

See also `https://github.com/Singular/Singular/wiki/Step-by-Step-Installation-Instructions-for-Singular` which includes instructions adapted for debian and fedora based systems.

8.7 Windows installation instructions

Singular relies on Cygwin as its environment under Windows. There is a 32bit and a 64bit version of Cygwin.

`https://www.singular.uni-kl.de/index.php/singular-download/install-windows.html`

8.8 Macintosh installation instructions

Installation of the provided binaries `https://www.singular.uni-kl.de/index.php/singular-download/install-os-x.html`

If your Mac refuses to open Singular because of an `"unidentified developer"`: Open System Preferences. Go to the Security & Privacy tab. Click on the lock and enter your password so you can make changes. Change the setting for 'Allow apps downloaded from' to 'App Store and identified developers'.

You may also check `https://support.apple.com/en-en/guide/mac-help/mh40616/mac`

9 Index

Table of Contents