Singular

A Computer Algebra System for Polynomial Computations

Manual

Version 3-1-1, Feb 2010

SINGULAR is created and its development is directed and coordinated by
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

Fachbereich Mathematik
Zentrum für Computeralgebra
Universität Kaiserslautern
D-67653 Kaiserslautern

Short Contents

1 Preface

SINGULAR version 3-1-1
University of Kaiserslautern
Department of Mathematics and Centre for Computer Algebra
Authors: G.-M. Greuel, G. Pfister, H. Schoenemann

NOTICE

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
`http://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 `http://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
`http://gmplib.org`

MP
: © Gray/Kajler/Wang, Kent State University
`http://www.symbolicnet.org/areas/protocols/mp.html`

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

Singular-Factory
© Greuel/Stobbe, University of Kaiserslautern:
ftp://www.mathematik.uni-kl.de/pub/Math/Singular/Factory

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

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

Cygwin http://www.cygwin.com/

Xemacs http://www.xemacs.org

Some external programs are optional:

4ti2 (used by sing4ti2.lib, see Section D.4.27 [sing4ti2_lib], page 875)
http://www.4ti2.de/

gfan (used by tropical.lib, see Section D.12.2 [tropical_lib], page 1210)
http://www.math.tu-berlin.de/~jensen/software/gfan/gfan.html

graphviz (used by resgraph.lib, see Section D.8.3 [resgraph_lib], page 1069)
http://www.graphviz.org/

normaliz (used by normaliz.lib, see Section D.4.16 [normaliz_lib], page 760)
© Winfried Bruns and Bogdan Ichim
http://www.mathematik.uni-osnabrueck.de/normaliz/

polymake (used by polymake.lib, see Section D.12.1 [polymake_lib], page 1196)
© Ewgenij Gawrilow and Michael Joswig
http://www.polymake.de/

surf (used by surf.lib, see Section D.8.4 [surf_lib], page 1070)
© Stephan Endrass
http://surf.sf.net

surfer (used by surf.lib, see Section D.8.4 [surf_lib], page 1070)
http://www.imaginary2008.de/surfer.php

surfex (used by surfex.lib, see Section D.8.5 [surfex_lib], page 1072)
© Oliver Labs (2001-2008), Stephan Holzer (2004-2005)
http://surfex.AlgebraicSurface.net

TOPCOM (used by polymake.lib, see Section D.12.1 [polymake_lib], page 1196)
© 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, Universit 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, Grgoire 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: Olaf Bachmann, Thomas Bauer, Markus Becker, Michael Cuntz, Kai Dehmann, Marcin Dumnicki, Stephan Endraß, Vladimir Gerdt, Philippe Gimenez, Christian Gorzel, Hubert Grassmann, Agnes Heydtmann, Dietmar Hillebrand, Tobias Hirsch, Manuel Kauers, Simon King, Anen Lakhal, Martin Lamm, Francisco Javier Lobillo, Michael Meßollen, Andrea Mindnich, Jorge Martin Morales, Thomas Nüßler, Wolfgang Neumann, Markus Perling, Wilfried Pohl, Carlos Rabelo, Alfredo Sanchez-Navarro, Jens Schmidt, Thomas Siebert, Rüdiger Stobbe, Henrik Strohmayer, Christian Stussak, Imade Sulandra, Christine Theis, Enrique Tobis, Alberto Vigneron-Tenorio, Moritz Wenk, Eric Westenberger, Tim Wichmann, Oliver Wienand, Denis Yanovich and Oleksandr Yena.

We should like to acknowledge the financial support given by the Volkswagen-Stiftung, the Deutsche Forschungsgemeinschaft and the Stiftung für Innovation des Landes Rheinland-Pfalz to the SINGULAR project.

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 3-1-1, see Section 8.1 [News and changes], page 1233.

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 434 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 29.
- An overview of the algorithms implemented in the kernel of SINGULAR is given in Section 3.4 [Implemented algorithms], page 35.
- In Section 3.5 [The SINGULAR language], page 39, 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 46, 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 49, Section 3.8 [Libraries], page 53, and Section 3.10 [Debugging tools], page 66.

Chapter 4 [Data types], page 70, 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 127, 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 254, 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 507, 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 525, 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 4.5 [intmat], page 82, 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 35), 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 35), 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 35), 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;
↦ //   characteristic : 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 29.

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 258 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 5.1.98 [option], page 192, 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 471, 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`:

```
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 `help "all.lib";` lists all libraries together with a one-line explanation.

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 501, and Appendix C [Mathematical background], page 507).

```
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 4.9 [map], page 98.

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 501).

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 $\mathrm{ann}(M) = \{a \mid aM = 0\}$ which is, by definition of M, the same as $\{a \mid ar^3 \in U\}$. Hence we have to calculate the quotient $U : r^3$. 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 24 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 21, Section 3.2.2 [Running SINGULAR under Emacs], page 24.

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 5.1.45 [help], page 157, 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`),
`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 5.1.137 [system], page 227)

```
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 explicitely 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 21) 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 netscape,
// 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 explicitely 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 `$RootDir/html` – see Section 3.8.1 [Loading a library], page 53 for more info on `$RootDir`) *and* the (command-line) option `--allow-net` has *explicitely* been set (see Section 3.1.6 [Command line options], page 19 and Section 5.1.137 [system], page 227 for more info on setting values of command-line options), then it dispatches the html pages from `http://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.hlp` which will be looked for at `$RootDir/info/singular.hlp` (see Section 3.8.1 [Loading a library], page 53 for more info on `$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 particuliar, if SINGULAR is able to load that library and if the environment variable `SINGULARHIST` is set and has a name of a valid file as value, then the input history is stored across sessions using this file. Otherwise, i.e., if the environment variable `SINGULARHIST` is not set, then 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 3.10.2 [Source code debugger], page 66.

`-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 248.

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

`--allow-net`
Allow the `netscape` and `html` help 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 `html` (Windows only), `netscape`, `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 `netscape` 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 21.

`--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 21.

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

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

`-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, --verbose`
: 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.

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 250), resp. `rtimer` (see Section 5.3.10 [rtimer], page 253) , 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.

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

`-b, --batch`
: Run in MP 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 MP format. See Section 4.7.5.2 [MPtcp links], page 92.

`--MPport=PORT`
: Use `PORT` as default port number for MP connections (whenever not further specified). This option is mandatory when the `--batch` option is given. See Section 4.7.5.2 [MPtcp links], page 92.

`--MPhost=HOST`
: Use `HOST` as default host for MP connections (whenever not further specified). This option is mandatory when the `--batch` option is given. See Section 4.7.5.2 [MPtcp links], page 92.

Finally, the following options are only available when running `ESingular` (see Section 3.2.2 [Running SINGULAR under Emacs], page 24 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 5.1.137 [system], page 227.

```
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 5.1.137 [system], page 227.

```
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 3.8.1 [Loading a library], page 53 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 approriate 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 (including Windows 95/98/NT, but excluding the Macintosh), from either http://www.gnu.org/software/emacs/emacs.html (Emacs), from http://www.xemacs.org (XEmacs), or from our ftp site at ftp://www.mathematik.uni-kl.de/pub/Math/Singular/utils/. 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 24 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. Under Windows, the full SINGULAR installation comes with (x)emacs and will create an ESingular startup icon on your desktop

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.

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 21). 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 let's 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 33, Appendix C [Mathematical background], page 507.

The coefficient field of the rings may be

1. the field of rational numbers Q ,
2. finite fields Z/p, p a prime ≤ 2147483629,
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,
9. finite rings Z/m with $m \in Z$.

In case of coefficient rings, which are not fields, only the following functions are guaranteed to work:

- basic polynomial arithmetic, i.e. addition, multiplication, division, exponentation
- std, i.e. computing standard bases
- 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 5.2.10 [keepring], page 244, Section 5.1.123 [setring], page 213.

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 43, Section 5.1.32 [fetch], page 148, Section 5.1.50 [imap], page 161, Section 4.9 [map], page 98.

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 49, Section 5.2.10 [keepring], page 244.

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 502.

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 r;
ring r = 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;
```

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

```
ring r = 0,(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 r = 7,(x,y,z),ds;
ring r = 10,(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 r = 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 r = 0,(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 r = 0,(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 `);`

Default: `32003,(x,y,z),(dp,C);`

Purpose: declares a ring and sets it as the current basering.

The coefficients are given by one of the following:

1. a non-negative int_expression less than or equal to 2147483629.
   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.
2. 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 248.
3. 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.
4. 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 exeption 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.
5. 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.
6. an expression_list with the name `integer`. This specifies the ring of integers.
7. an expression_list with the name `integer` and one subsequent int_expression. This specifies the ring of integers modulo the given int_expression.
8. 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 502.

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 5.1.32 [fetch], page 148).

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 5.1.115 [reduce], page 206).

Example:

```
  ring r=32003,(x,y),dp;
  poly f=x3+yx2+3y+4;
  qring q=std(maxideal(2));
  basering;
↦ //   characteristic : 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
```

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 501.

In a ring declaration, SINGULAR offers the following orderings:

1. Global orderings

   `lp` lexicographical ordering

   `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.

   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

   `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 `)`
   an extra weight vector `a(` intvec_expression `)` may precede any monomial ordering

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`, and `Ds` 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 501.

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 5.1.29 [facstd], page 146
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 5.1.33 [fglm], page 149
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 5.1.44 [groebner], page 155
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 5.1.88 [mstd], page 187
computes a standard basis and a minimal set of generators.

`std` Section 5.1.133 [std], page 223
computes a standard resp. Groebner basis.

`stdfglm` Section 5.1.134 [stdfglm], page 224
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 5.1.135 [stdhilb], page 225
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 5.1.16 [degree], page 139
computes the (Krull) dimension, codimension and the multiplicity.

The result is only displayed on the screen.

`dim` Section 5.1.20 [dim], page 141
computes the dimension of the ideal resp. module.

`highcorner` Section 5.1.46 [highcorner], page 158
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 5.1.47 [hilb], page 159
computes the first, and resp. or, second Hilbert series of an ideal resp. module.

`kbase` Section 5.1.60 [kbase], page 168
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 5.1.89 [mult], page 188
computes the degree of the monomial ideal resp. module generated by the leading monomials of the input.

`reduce` Section 5.1.115 [reduce], page 206
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 5.1.149 [vdim], page 234
computes the vector space dimension of a ring (resp. free module) modulo an ideal (resp. module).

Commands to compute resolutions

`res` Section 5.1.118 [res], page 209
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.

`lres` Section 5.1.74 [lres], page 177
computes a free resolution of an ideal or module with LaScala's method. The input needs to be homogeneous.

`mres` Section 5.1.87 [mres], page 186
computes a minimal free resolution of an ideal or module with the Syzygy method.

`sres` Section 5.1.131 [sres], page 221
computes a free resolution of an ideal or module with Schreyer's method. The input has to be a standard basis.

`nres` Section 5.1.94 [nres], page 190
computes a free resolution of an ideal or module with the standard basis method.

`syz` Section 5.1.138 [syz], page 229
computes the first Syzygy (i.e., the module of relations of the given generators).

Further processing of resolutions

`betti` Section 5.1.3 [betti], page 130
computes the graded Betti numbers of a module from a free resolution.

`minres` Section 5.1.82 [minres], page 184
minimizes a free resolution of an ideal or module.

`regularity` Section 5.1.116 [regularity], page 208
computes the regularity of a homogeneous ideal resp. module from a given minimal free resolution.

Processing of polynomials

`char_series` Section 5.1.5 [char_series], page 132
computes characteristic sets of polynomial ideals.

`extgcd` Section 5.1.28 [extgcd], page 146
computes the extended gcd of two polynomials.

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

`factorize`
: Section 5.1.30 [factorize], page 147
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 5.1.41 [gcd], page 154
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 5.1.120 [resultant], page 211
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 5.1.145 [vandermonde], page 232
interpolates a polynomial from its values at several points

Matrix computations

`bareiss`
: Section 5.1.2 [bareiss], page 128
implements sparse Gauss-Bareiss method for elimination (matrix triangularization) in arbitrary integral domains.

`det`
: Section 5.1.18 [det], page 140
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 5.1.81 [minor], page 182
computes all minors (=subdeterminants) of a given size for a matrix.

Numeric computations

`laguerre`
: Section 5.1.65 [laguerre], page 171
computes all (complex) roots of a univariate polynomial

`uressolve`
: Section 5.1.144 [uressolve], page 232
finds all roots of a 0-dimensional ideal with multivariate resultants

Controlling computations

`option` Section 5.1.98 [option], page 192
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 49, and Section 3.8 [Libraries], page 53, 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 254.

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 70, and its subsections (e.g., Section 4.14.2 [poly expressions], page 113), 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 39.
- "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 237, 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 44.
2. 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 44.
3. 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 42, for more information on declarations. See Chapter 4 [Data types], page 70, 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 44.

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 127. 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 5.1.107 [print], page 200.

```
  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 258, for the difference between the division operators `/` and `div`.
`%`	modulo operator (`mod` is an alias to `%`)
`^` 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 5.1.35 [filecmd], page 150.

`!`	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 39).

See Chapter 4 [Data types], page 70, for a description of SINGULAR's data types. See Section 5.1.142 [typeof], page 231, 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 5.1.141 [type], page 231).

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 **
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. 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.

```
ring R;
int n=3;
ideal j(3);
ideal j(n);     // is equivalent to the above
↦ // ** redefining j(3) **
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;
↦ //   characteristic : 0
↦ //   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
```

Names must not coincide with reserved names (keywords). Type `reservedName();` to get a list of the reserved names. See Section 5.1.119 [reservedName], page 210. Names should not interfere with names of ring variables or, more generally, with monomials. See Section 6.5 [Identifier resolution], page 260.
The command `listvar` provides a list of the names in use (see Section 5.1.73 [listvar], page 176).

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 5.1.142 [typeof], page 231, Section 5.1.1 [attrib], page 127):

```
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 4.9 [map], page 98, Section 5.1.32 [fetch], page 148, Section 5.1.50 [imap], page 161.

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. `int` ↦ `ideal`
2. `poly` ↦ `ideal`
3. `intvec` ↦ `intmat`
4. `int` ↦ `intvec`
5. `int` ↦ `intmat`
6. `string` ↦ `link`
7. `resolution` ↦ `list`
8. `ideal` ↦ `matrix`
9. `int` ↦ `matrix`
10. `intmat` ↦ `matrix`
11. `intvec` ↦ `matrix`
12. `module` ↦ `matrix`
13. `number` ↦ `matrix`
14. `poly` ↦ `matrix`
15. `vector` ↦ `matrix`
16. `ideal` ↦ `module`
17. `matrix` ↦ `module`
18. `vector` ↦ `module`
19. `int` ↦ `number`

20. `int` ↦ `poly`
21. `number` ↦ `poly`
22. `string` ↦ `proc`
23. `list` ↦ `resolution`
24. `int` ↦ `vector` (i ↦ i*`gen(1)`)
25. `poly` ↦ `vector` (p ↦ p*`gen(1)`)

Type casting

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

Possible type casts are:

to	from
`ideal`	expression lists of `int`, `number`, `poly`
`ideal`	`int`, `matrix`, `module`, `number`, `poly`, `vector`
`int`	`number`, `poly`
`intvec`	expression lists of `int`, `intmat`
`intmat`	`intvec` (see Section 4.5.3 [intmat type cast], page 83)
`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 4.10.3 [matrix type cast], page 102)
`module`	expression lists of `int`, `number`, `poly`, `vector`
`module`	`ideal`, `matrix`, `vector`
`number`	`poly`
`poly`	`int`, `number`
`string`	any type (see Section 4.19.3 [string type cast], page 122)

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 5.2.8 [if], page 242, Section 5.2.4 [else], page 239). `for` and `while` loops are available for a repeated execution of blocks (see Section 5.2.7 [for], page 242, Section 5.2.14 [while], page 246). In procedure definitions, the main part and the example section are blocks as well(see Section 4.15 [proc], page 116).

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 5.1.70 [LIB], page 174. For executing program scripts, see Section 5.1.35 [filecmd], page 150.

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 5.1.84 [monitor], page 185).

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 5.1.153 [write], page 236, Section 5.1.114 [read], page 206, Section 5.1.22 [dump], page 142, Section 5.1.43 [getdump], page 155.

Example:

```
ring r; poly p = x+y;
dump("MPfile:w test.mp");   // dump the session to the file test.mp
kill r;                     // kill the basering
listvar();                  // no output after killing the ring
getdump("MPfile:r test.mp");// 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. Except for MPtcp links, 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 5.1.97 [open], page 192, Section 5.1.9 [close], page 133).

Links have various properties which can be queried using the `status` function (see Section 5.1.132 [status], page 222).

Example:

```
  link l = "MPtcp:fork";
  l;
↦ // type : MPtcp
↦ // mode : fork
↦ // name :
↦ // open : no
↦ // read : not ready
↦ // write: not ready
  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 5.1.27 [execute], page 145).

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 4.7.4 [ASCII links], page 89, 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
```

MPfile links

Data is stored in the binary MP format. Read and write access is very fast compared to ASCII links. All data (including such data that cannot be converted to a string) can be written to an MPfile link. Reading from an MPfile link returns the written expressions (i.e., not a string, in general).

MPfile links should primarily be used for storing large amounts of data (like dumps of the content of an entire Singular session), and if the data to be stored cannot be easily converted from or to a string (like rings, or maps).

MPfile links are implemented on Unix-like operating systems only.

See Section 4.7.5.1 [MPfile links], page 91, for more information.

Example:

```
ring r;
// (over)write MPfile test.mp, link is specified as string
write("MPfile:w test.mp", x+y);
kill r;
def p = read("MPfile:r test.mp");
typeof(p); p;
↦ poly
↦ x+y
```

MPtcp 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 binary MP format. Reading from an MPtcp link returns the written expressions (i.e., not a string, in general).

MPtcp links should primarily be used for communicating with other programs or for parallel computations (see, for example, Section A.1.8 [Parallelization with MPtcp links], page 443).

MPtcp links are implemented on Unix-like operating systems only.

See Section 4.7.5.2 [MPtcp links], page 92, for more information.

Example:

```
ring r;
link l = "MPtcp:launch"; // 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 4.7.6 [DBM links], page 94, 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 of the SINGULAR language. They are used to extend the set of commands by user defined commands. Procedures are defined by either typing them on the command line or by loading them from a so-called library file with the LIB command, see Section 5.1.70 [LIB], page 174. Procedures are invoked like normal built-in commands, i.e., by typing their name followed by the list of arguments in parentheses. The invocation then executes the sequence of commands stored in the specified procedure. All defined procedures can be displayed by the command `listvar(proc);`.

See Section 3.9.1 [Procedures in a library], page 55.

3.7.1 Procedure definition

Syntax:

```
[static] proc proc_name [parameter_list]
["help_text"]
{
  procedure_body
}
[example
{
  sequence_of_commands;
}]
```

Purpose: defines a new function, the `proc` proc_name, with the additional information help_text, which is written to the screen by `help proc_name;` and the `example` section which is executed by `example proc_name;`.
The help_text, the parameter_list, and the example section are optional. The default for a parameter_list is `(list #)`, see Section 3.7.3 [Parameter list], page 51. The help and example sections are ignored if the procedure is defined interactively, i.e., if it was not loaded from a file by a command as in Section 5.2.11 [load], page 245.
Specifying `static` in front of the proc-definition (in a library file) makes this procedure local to the library, i.e., accessible only for the other procedures in the same library, but not for the users. So there is no reason anymore to define a procedure within another one (it just makes debugging harder).

Example of an interactive procedure definition

```
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, the library definition:

```
// 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 the procedures defined there:

```
  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 /home/hannes/singular/doc/singular.\
   idx
↦ // proc tab from lib sample.lib
↦ proc tab (int n)
↦ USAGE:    tab(n);  (n integer)
↦ RETURNS:  string of n space tabs
↦ EXAMPLE:  example tab; shows an example
```

Guidelines for the help text of a procedure

There are no enforced rules on the format of the help section of a procedure.

Nevertheless, we recommend that the help text of a procedure should contain information about the usage, purpose, return values and generated objects. Particular assumptions or limitations should be listed. It should also be mentioned if global objects are generated or manipulated.

The help text of procedures contained in libraries of the SINGULAR distribution should furthermore comply with certain rules as explained in Section 3.9.5 [The help string of procedures], page 61.

3.7.2 Names in procedures

All variables are local to the procedure. They are defined in the package in which the procedure is defined. Locally defined variables cannot interfere with names in other procedures and are automatically deleted after leaving the procedure.

Internally, local variables are stored using the nesting level. A variable is said to have nesting level 1, if it is local to a procedure that was called interactively, nesting level 2, if it is local to a procedure that was called by a procedure of nesting level 1 etc. `listvar()` also displays the nesting level, nesting level 0 is used for global objects (see Section 5.1.73 [listvar], page 176).

To keep local variables after leaving the procedure, they have to be exported (i.e. made known) to some higher level or to some package by a command like `export` or `exportto` (see Section 5.2.5 [export], page 239, see Section 5.2.6 [exportto], page 240, see Section 5.2.9 [importfrom], page 242; see also see Section 4.13 [package], page 111).

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`.

3.7.3 Parameter list

Syntax: `( )`
`( parameter_definition )`

Purpose: defines the number, type and names of the arguments to a `proc`.
The parameter_list is optional. The default for a parameter_list is `(list #)` which means the arguments are referenced by `#[1]`, `#[2]`, etc.
If a procedure has optional parameters (i.e. `(list #)` appears in the declaration), the attribute `default_arg` gives the default values for this optional arguments. (This provides 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, e.g.

```
proc x6 (r)
{
... // this is correct even if the parameter is a ring
}

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

3.7.4 Procedure commands

Some commands only make sense inside a procedure, since they make objects known to the nesting level from which the procedure was called or to all nesting levels.

See Section 5.2.5 [export], page 239; Section 5.2.10 [keepring], page 244; Section 5.2.13 [return], page 246.

3.8 Libraries

A library is a collection of SINGULAR procedures in a file.
SINGULAR reads a library with the commands `load` and `LIB`. General information about the library is displayed by the command `help libname_lib`. After loading the library, its procedures can be used like any built-in SINGULAR function.

To have the full functionality of a built-in function, libraries have to comply with the set of syntax rules described below.

Furthermore, libraries which are to be included in the SINGULAR distribution, have to comply with certain rules as explained in Section 3.9 [Guidelines for writing a library], page 55.

3.8.1 Loading a library

Libraries can be loaded with the `LIB` or the `load` command (see Section 5.2.11 [load], page 245):

Syntax: `LIB` string_expression `;`

Type: none

Purpose: reads a library of procedures from a file. If the given filename does not start with `.` or `/` and cannot be located in the current directory, each directory contained in the `SearchPath` is searched for 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/Singular/` on a Unix platform, `\Singular\` on a Windows 95/98/NT/XP/Vista platform.
3. all directories which do not exist are removed from the `SearchPath`.

For setting environment variables see Section 5.1.137 [system], page 227, 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.

Only the names of the procedures in the library are loaded, the body of each procedure is only 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. More precisely, option `-q` (and likewise `--quiet`) unsets option `loadLib` to inhibit monitoring of library loading (see Section 5.1.98 [option], page 192).

All loaded libraries are displayed by the `listvar(package);` command:

```
  option(loadLib);   // show loading of libraries;
                     // standard.lib is loaded
  listvar(package);
↦ // Standard             [0]  package (S,standard.lib)
↦ // Top                  [0]  package (N)
                     // the names of the procedures of inout.lib
  LIB "inout.lib";   // are now known to Singular
↦ // ** loaded inout.lib (12541,2010-02-09)
  listvar(package);
↦ // Inout                [0]  package (S,inout.lib)
↦ // Standard             [0]  package (S,standard.lib)
↦ // Top                  [0]  package (N)
```

See Section 3.1.6 [Command line options], page 19; Section 5.1.70 [LIB], page 174; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 525; Section 4.15 [proc], page 116; Section D.1 [standard_lib], page 525; Section 4.19 [string], page 120; Section 5.1.137 [system], page 227.

3.8.2 Format of a library

A library file can contain comments, a category-, info- and version-string definition, `LIB` commands, `proc` commands and `proc` commands with `example` and `help` sections, i.e., the following keywords are allowed: `category`, `info`, `version`, `LIB`, `/* ... */`, `//`, `[static] proc`. Anything else is not recognized by the parser of SINGULAR and leads to an error message while loading the library. If an error occurs, loading is aborted and an error message is displayed, specifying the type of error and the line where it was detected.

The first line of the library should start with the string `// Singular-library`.

The category-, info- and version-string are defined as follows:

Syntax: `info` = string_constant `;`

Purpose: defines the general help for the library. This text is displayed on `help libname_lib;`.

Example:

```
info="
      This could be the general help of a library.
     Quotes must be escaped with a \ such as \"
";
```

Note: In the info-string the characters `\` and `"` must be preceded by a `\` (escaped). It is recommended that the info string is placed at the head of a library file and contains general information about the library as well as a listing of all procedures available to the users (with a one line description of each procedure).

Although there is no enforced format of the info string of a library, we recommend that you follow certain rules as explained in Section 3.9.4 [The help string of a library], page 61.

Syntax: `version` = string_constant `;`

Purpose: defines the version number for the library. It is displayed when the library is loaded.

Example:

```
version="$Id: sample.lib,v 1.2 1998/05/07 singular Exp $";
version="some version string";
```

Note: It is common practice to simply define the version string to be `"$Id:$"` and let a version control system expand it.

Syntax: `category` = string_constant ;

Purpose: defines the category for the library.

Example:

```
category="Utilities";
```

Note: reserved for sorting the libraries into categories.

3.9 Guidelines for writing a library

Although there are very few enforced rules on how SINGULAR libraries should be written (see Section 3.8 [Libraries], page 53), it is recommended that the libraries comply with the guidelines explained in this section, so that debugging and understanding are made easier.

Note: For libraries which are to be included in the SINGULAR distribution, the following guidelines are mandatory.

3.9.1 Procedures in a library

In this section we list miscellaneous recommendations on how procedures contained in a library should be implemented.

1. The info- and version-string should appear at the beginning of the library, before the first procedure definition.
2. The info-string should have the format as explained in Section 3.9.4 [The help string of a library], page 61.
3. Each procedure which should not be accessible by users should be declared `static`.
4. Each procedure which is not declared `static` should have a help and example section as explained in Section 3.7.1 [Procedure definition], page 49.
   Such procedures should furthermore carefully check any assumptions made about their input (like the type of list elements), and, if necessary, report an error using the function Section 5.1.25 [ERROR], page 145.
5. 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 discourraged. 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.
6. No procedures should be defined within the body of another procedure.
7. If the value of the reserved variable `printlevel` (see Section 5.3.6 [printlevel], page 249) is greater than 0 then interactive user-input, i.e., the usage of functions like `pause("..")` or `read("");` (see Section 5.1.114 [read], page 206), may be requested.
8. If the value of the reserved variable `printlevel` (see Section 5.3.6 [printlevel], page 249) is 0 then interactive user-input, i.e., the usage of functions like `pause("..")` or `read("");` (see Section 5.1.114 [read], page 206), may **not** be requested. Instead, an error (using the function Section 5.1.25 [ERROR], page 145) should be reported together with the recommendation on increasing the value of the reserved variable `printlevel`.
9. It is often useful for a procedure to print out comments, either for explaining results or for displaying intermediate computations. However, if this procedure is called by another procedure, such comments are confusing and disturbing in most cases.

SINGULAR offers an elegant solution, which requires the usage of the SINGULAR function Section 5.1.13 [dbprint], page 137 and the reserved variables Section 5.3.6 [printlevel], page 249, and Section 5.3.11 [voice], page 253 (`voice` counts the nesting of procedures; It has the value 1 on the top level, 2 inside the first procedure etc.; `printlevel` has the value 0 by default, but can be set to any integer value by the user).

For example, if the following procedure `Test` is called directly from the top level then '`comment1`' is displayed (i.e., printed out) but not '`comment2`'; and 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 that `Test` is called from other procedures, with a nesting level up to k (resp. k-1).

Note furthermore, that the example part of a procedure behaves in this respect like a procedure (i.e., the value of `voice` is 1). Therefore, the command `printlevel=1;` is necessary for '`comment1`' to be displayed on `example Test;`. However, since printlevel is a global variable, it should be reset to the old value at the end of the example part.

```
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
}
```

3.9.2 Documentation of a library

The typesetting language in which the SINGULAR documentation is written is `texinfo`. Based on various tools, `info`, `dvi`, `ps`, and `html` versions of the `texinfo` documentation are generated.

Starting with SINGULAR version 1-3, the `texinfo` documentation of all libraries of the SINGULAR distribution is generated automatically from their source code.

More precisely, for each library,

- the info string of the library is parsed and typeset as explained in Section 3.9.3 [Typesetting of help strings], page 58.
- the help string of each procedure listed in the `PROCEDURE:` section of the library info string is parsed and typeset as explained in Section 3.9.3 [Typesetting of help strings], page 58.
- the example of each procedure listed in the `PROCEDURE:` section of the library info string is computed and its output is included into the documentation.

For a uniform look-and-feel of the library documentation, library developers should

- follow the recommendation of Section 3.9.4 [The help string of a library], page 61 and Section 3.9.5 [The help string of procedures], page 61.
- consult the source code of libraries like `template.lib` (see Section 3.9.6 [template_lib], page 62) for examples on how library documentations are written.
- make sure that each procedure listed in the `PROCEDURE:` section of the library info string has a help string and an example section.
- not use interactive functions like `pause("..")` or `read("");` (see Section 5.1.114 [read], page 206) and should limit the length of input lines to 60 characters in the example section of procedures.
- try to avoid time-consuming operations in loops: especially do not forget automatic type conversions.
- carefully check the generated documentation of their libraries in its various formats using the `lib2doc` (see Section 3.9.2.1 [lib2doc], page 57) utility.

3.9.2.1 lib2doc

`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://www.mathematik.uni-kl.de/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)

```
netscape mylib.html
```

`make clean`
: Deletes all generated files.

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

3.9.3 Typesetting of help strings

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

For optimal typesetting results, the guidelines for writing libraries and procedures should be followed, and the following points should be kept in mind:

- If a help string starts with an `@` sign, then no parsing is done, and the help string is assumed to be already in the texinfo format.
- help strings are typeset within a `@table @asis` environment (which is similar to a latex `description` environment).
- If a line starts with only uppercase words and contains 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

  `EXAMPLE` then this item and its content is ignored.

  `SEE ALSO` then the content of the item is assumed to be 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` (or, `KEYPHRASES`)
  : 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 (N.B. the name of a procedure/library is automatically added to the index keys).

  `PROCEDURES`
  : then the content of the item is assumed to be a summary description of the procedures contained in the library. Separate texinfo nodes (subsections in printed documents) are created exclusively from the help strings of such procedures appearing in the summary description of a library.

  `LIBRARY` then the content of the item is assumed to be a one-line description of a library. If this one-line description consist of only uppercase characters, then it is typeset in all lowercase characters in the manual (otherwise it is left as is).
- For the content of an item, the following texinfo markup elements are recognized (and, their content not further manipulated):

  `@*` to enforce a line-break.

  Example: `old line @* new line`
  ↦

old line
new line

`@ref{...}`

References to other parts of the SINGULAR manual can be set using one of the following `@ref{node}` constructs. Notice that `node` must be the name of a section of the SINGULAR manual. In particular, it may be a name of a function, library or library procedure.

`@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 parenthesis.

Example: `@xref{Tropical Storms}, 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{..}`

for typesetting of small (i.e., which do not expand over multiple lines) mathematical expressions in LaTeX math-mode syntax.

Example: `@math{\alpha}`
↦
α

Note: Mathematical expressions inside `@math{..}` must not contain curly parenthesis and the "at" sign, i.e., may not contain `{,},@`.

`@code{..}`

for typesetting of small (i.e., which do not go over multiple lines) strings in typewriter font.

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

Note: The string inside `@code{..}` must not contain curly parenthesis and the "at" sign, i.e., must not contain `{,},@`.

`@example ...`

`@end example`

for pre-formatted text which is indented and typeset in typewriter font.

Example:

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

↦

```
before example
    in                 example
    notice extra identation  and
    escape of special characters like {,},@
after example
```

Note: The characters `{,},@` have to be escaped by an `@` sign inside an `@example` environment.

`@format` ...

`@end format`

for pre-formatted text which is not indented and typeset in normal font.

Example:

```
before format
@format
in              format
no extra identation but still
escape of special characters like @{,@},@@
@end format
after format
```

↦

```
before format
in              format
no extra identation  but still
escape of special characters like {,},@
after format
```

Note: The characters `{,},@` have to be escaped by an `@` sign inside an `@example` environment.

`@texinfo` ...

`@end texinfo`

for text which is written in pure 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.

See also Section 3.9.6 [template_lib], page 62 for an examples of the typesetting rules explained here.

3.9.4 The help string of a library

The help (or, info) string of a library should have the following format:

```
info="
LIBRARY: <library_name> <one line description of the content>
AUTHOR:  <name, and email address of author>
[SEE ALSO: <comma-separated words of cross references>]
[KEYWORDS: <semicolon-separated phrases of index keys>]
PROCEDURES:
  <procedure1>;     <one line description of the purpose>
   .
   .
  <procedureN>;     <one line description of the purpose>
";
```

Only such procedures should be listed in the `PROCEDURE` section which are not `static` and which have a help and example section.

The purpose of the one line procedure descriptions is not to give a short help for the procedure, but to help the user decide what procedure might be the right one for the job. Details can then be found in the help section of each procedure. Therefore, parameters may be omitted or abbreviated if necessary. If this description consists of only upper-case characters, then it will be typeset in all lowercase characters in the manual.

Please note, that the section `PROCEDURES:` must be the last section and the only section to list procedures.

For more information, see Section 3.9.3 [Typesetting of help strings], page 58. For an example, see Section 3.9.6 [template_lib], page 62.

3.9.5 The help string of procedures

The help string of a procedure should have the following format:

```
USAGE:    <proc_name>(<parameters>);   <explanation of parameters>
[CREATE:  <description of created objects which are not returned>]
RETURN:   <description of the purpose and return value>
[NOTE:    <particular assumptions or limitations, details>]
[SEE ALSO: <comma-separated names of related procedures/cross references>]
[KEYWORDS: <semicolon-separated phrases of index keys>]
EXAMPLE:  example <proc_name>; shows an example
```

Further arbitrary items (like `THEORY:`, or `BACKGROUND:`) are recognized, as well, but should be used diligently.

Remember that help strings are formatted as explained in Section 3.9.3 [Typesetting of help strings], page 58. In particular, descriptions may contain the texinfo markup elements `@*`, `@math{..}`,

`@code{..}`, `@example`, `@format`, `@texinfo` to better control their typesetting. See Section 3.9.6.3 [msum], page 65, Section 3.9.6.1 [mdouble], page 64, Section 3.9.6.2 [mtripple], page 64 for examples.

3.9.6 template_lib

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

```
////////////////////////////////////////////////////////////////////
// version string automatically expanded by CVS

version="$Id: template.lib 12231 2009-11-02 10:12:22Z hannes $";
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, Guidelines for writing a library,
           Typesetting of help strings

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

PROCEDURES:
  mdouble(int)           return double of int argument
  mtripple(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
          mdouble(i): no new line
SEE ALSO: msum, mtripple, Typesetting of help strings
KEYWORDS: procedure, ASCII help
EXAMPLE:  example mdouble; shows an example"
{
  return (i + i);
}
example
{ "EXAMPLE:"; echo = 2;
  mdouble(0);
  mdouble(-1);
}
////////////////////////////////////////////////////////////////////
proc mtripple(int i)
"@c we do texinfo here
@table @asis
@item @strong{Usage:}
@code{mtripple(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 (like
@math{\alpha, \beta}).
@*It 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 strings}
@c ref
"
{
  return (i + i + i);
}
example
{ "EXAMPLE:"; echo = 2;
  mtripple(0);
  mtripple(-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 a @ref constructs for references (like @pxref{mtripple})
         @* Use @code  for typewriter font (like @code{i_1})
         @* Use @math  for simple math mode typesetting (like @math{i_1}).
         @* Note: No parenthesis like } are allowed inside @math and @code
         @* Use @example for indented preformatted text typeset in typewriter
         font like
@example
         this  --> that
@end example
        Use @format  for preformatted text typeset 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
        Notice that
        automatic linebreaking         is still in affect (like on this line).
SEE ALSO: mdouble, mtripple, Typesetting of help 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);
}
```

After typesetting, the library appears in the document as follows (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 3.9 [Guidelines for writing a library], page 55; Section 3.9.3 [Typesetting of help strings], page 58; Section D.1 [standard_lib], page 525.

3.9.6.1 mdouble

Procedure from library `template.lib` (see Section 3.9.6 [template_lib], page 62).

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 mdouble(i): no new line

Example:

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

See also: Section 3.9.3 [Typesetting of help strings], page 58; Section 3.9.6.3 [msum], page 65; Section 3.9.6.2 [mtripple], page 64.

3.9.6.2 mtripple

Procedure from library `template.lib` (see Section 3.9.6 [template_lib], page 62).

Usage: `mtripple(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 (like

α, β).
It also gives more control over the layout, but is, admittingly, more cumbersome to write.

See also:

Example:

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

3.9.6.3 msum

Procedure from library `template.lib` (see Section 3.9.6 [template_lib], page 62).

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 a @ref constructs for references (like see Section 3.9.6.2 [mtripple], page 64)
Use @code for typewriter font (like `i_1`)
Use @math for simple math mode typesetting (like i_1).
Note: No parenthesis like `}` are allowed inside @math and @code
Use @example for indented preformatted text typeset in typewriter font like

```
this  --> that
```

Use @format for preformatted text typeset in normal font

this –> that

Use @texinfo for text in pure texinfo

↦ $i_{1,1}$

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

Example:

```
LIB "template.lib";
msum();
↦ 0
msum(4);
↦ 4
msum(1,2,3,4);
↦ 10
```

See also: Section 3.9.3 [Typesetting of help strings], page 58; Section 3.9.6.1 [mdouble], page 64; Section 3.9.6.2 [mtripple], page 64.

3.10 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⟩ .

3.10.1 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 251.

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)
```

3.10.2 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 5.2.2 [breakpoint], page 238).
These breakpoints can be cleared with the command `d breakpoint_no` from within the debugger or with `breakpoint(` proc_name `,-1);`.

3.10.3 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 5.2.15 [~], page 247.

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
```

3.10.4 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 5.1.13 [dbprint], page 137; Section 5.3.11 [voice], page 253.

3.10.5 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 occured 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";`
```

3.11 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, but in contrast to the case of a library, the name of the `package` is not derived from the filename, but is hardcoded in the dynamic module. In particular, general information about it 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 can be found at `http://www.singular.uni-kl.de/DynMod.ps`.

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 39.

4.1 bigint

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

4.1.1 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
```

4.1.2 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
↦ // ** int overflow(*), result may be wrong
↦ // ** int overflow(*), result may be wrong
↦ // ** int overflow(*), result may be wrong
↦ -1875651584
// 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 44; Section 4.4 [int], page 77; Section 4.12 [number], page 108.

4.1.3 bigint operations

`+` addition

`-` negation or subtraction

`*` multiplication

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

`mod` integer modulo (the remainder of the division `div`), always non-negative

`^`, `**` 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;
↦ -2 1
```

4.1.4 bigint related functions

`gcd` greatest common divisor (see Section 5.1.41 [gcd], page 154)

`memory` memory usage (see Section 5.1.79 [memory], page 181)

See Section 5.1.79 [memory], page 181.

4.2 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;
↦ //   characteristic : 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.

4.2.1 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 5.1.142 [typeof], page 231.

4.3 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 5.1.126 [size], page 217, Section 5.1.92 [ncols], page 190.

4.3.1 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
```

4.3.2 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);
```

4.3.3 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 5.1.125 [simplify], page 216.

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
```

4.3.4 ideal related functions

`char_series`
: irreducible characteristic series (see Section 5.1.5 [char_series], page 132)

`coeffs` matrix of coefficients (see Section 5.1.11 [coeffs], page 134)

`contract` contraction by an ideal (see Section 5.1.12 [contract], page 136)

`diff` partial derivative (see Section 5.1.19 [diff], page 140)

`degree` multiplicity, dimension and codimension of the ideal of leading terms (see Section 5.1.16 [degree], page 139)

`dim` Krull dimension of basering modulo the ideal of leading terms (see Section 5.1.20 [dim], page 141)

`eliminate`
: elimination of variables (see Section 5.1.23 [eliminate], page 143)

`facstd` factorizing Groebner basis algorithm (see Section 5.1.29 [facstd], page 146)

`factorize`
: ideal of factors of a polynomial (see Section 5.1.30 [factorize], page 147)

`fglm` Groebner basis computation from a Groebner basis w.r.t. a different ordering (see Section 5.1.33 [fglm], page 149)

`finduni` computation of univariate polynomials lying in a zero dimensional ideal (see Section 5.1.37 [finduni], page 151)

`groebner` Groebner basis computation (a wrapper around `std,stdhilb,stdfglm`,...) (see Section 5.1.44 [groebner], page 155)

`highcorner`
: the smallest monomial not contained in the ideal. The ideal has to be zero-dimensional. (see Section 5.1.46 [highcorner], page 158)

`homog` homogenization with respect to a variable (see Section 5.1.48 [homog], page 160)

`hilb` Hilbert series of a standard basis (see Section 5.1.47 [hilb], page 159)

`indepSet` sets of independent variables of an ideal (see Section 5.1.52 [indepSet], page 162)

`interred` interreduction of an ideal (see Section 5.1.55 [interred], page 165)

`intersect`
: ideal intersection (see Section 5.1.56 [intersect], page 165)

`jacob` ideal of all partial derivatives resp. jacobian matrix (see Section 5.1.57 [jacob], page 166)

`jet` Taylor series up to a given order (see Section 5.1.59 [jet], page 167)

`kbase` vector space basis of basering modulo ideal of leading terms (see Section 5.1.60 [kbase], page 168)

`koszul` Koszul matrix (see Section 5.1.64 [koszul], page 170)

`lead` leading terms of a set of generators (see Section 5.1.66 [lead], page 172)

`lift` lift-matrix (see Section 5.1.71 [lift], page 174)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.72 [liftstd], page 175)

`lres` free resolution for homogeneous ideals (see Section 5.1.74 [lres], page 177)

`maxideal` power of the maximal ideal at 0 (see Section 5.1.78 [maxideal], page 181)

`minbase` minimal generating set of a homogeneous ideal, resp. module, or an ideal, resp. module, in a local ring (see Section 5.1.80 [minbase], page 182)

`minor` set of minors of a matrix (see Section 5.1.81 [minor], page 182)

`modulo` representation of $(h1 + h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 5.1.83 [modulo], page 185)

`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 5.1.87 [mres], page 186)

`mstd` standard basis and minimal generating set of an ideal (see Section 5.1.88 [mstd], page 187)

`mult` multiplicity, resp. degree, of the ideal of leading terms (see Section 5.1.89 [mult], page 188)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`nres` a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 5.1.94 [nres], page 190)

`preimage` preimage under a ring map (see Section 5.1.104 [preimage], page 198)

`qhweight` quasihomogeneous weights of an ideal (see Section 5.1.110 [qhweight], page 203)

`quotient` ideal quotient (see Section 5.1.112 [quotient], page 204)

`reduce` normalform with respect to a standard basis (see Section 5.1.115 [reduce], page 206)

`res` free resolution of an ideal resp. module but not changing the given ideal resp. module (see Section 5.1.118 [res], page 209)

`simplify` simplification of a set of polynomials (see Section 5.1.125 [simplify], page 216)

`size` number of non-zero generators (see Section 5.1.126 [size], page 217)

`slimgb` Groebner basis computation with slim technique (see Section 5.1.127 [slimgb], page 218)

`sortvec` permutation for sorting ideals resp. modules (see Section 5.1.128 [sortvec], page 219)

`sres` free resolution of a standard basis (see Section 5.1.131 [sres], page 221)

`std` standard basis computation (see Section 5.1.133 [std], page 223)

`stdfglm` standard basis computation with fglm technique (see Section 5.1.134 [stdfglm], page 224)

`stdhilb` Hilbert driven standard basis computation (see Section 5.1.135 [stdhilb], page 225)

`subst` substitution of a ring variable (see Section 5.1.136 [subst], page 226)

`syz` computation of the first syzygy module (see Section 5.1.138 [syz], page 229)

`vdim` vector space dimension of basering modulo ideal of leading terms (see Section 5.1.149 [vdim], page 234)

`weight` optimal weights (see Section 5.1.151 [weight], page 235)

4.4 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 5.2.7 [for], page 242), and to represent boolean values (FALSE is represented by 0, every other value means TRUE, see Section 4.4.5 [boolean expressions], page 80).

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 4.1 [bigint], page 70.

4.4.1 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
```

4.4.2 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 254). 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
↦ // ** int overflow(*), result may be wrong
↦ // ** int overflow(*), result may be wrong
↦ // ** int overflow(*), result may be wrong
↦ -1875651584
// using the type cast number for a greater allowed range
number(11)*13*17*100*200*2000*503*1111*222222;
↦ 12075748128684240000000
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 44; Section 4.12 [number], page 108.

4.4.3 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

`/` integer division (omitting the remainder), rounding toward 0

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

`%` integer modulo (the remainder of the division `/`)

`mod` integer modulo (the remainder of the division `div`), always non-negative

`^`, `**` 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 254.

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
  // 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 / 2, -5 mod 2, -5 % 2;
↦ -3 -2 1 -1
  1<2, 2<=2;
↦ 1 1
```

4.4.4 int related functions

`char` characteristic of the coefficient field of a ring (see Section 5.1.4 [char], page 131)

`deg` degree of a polynomial resp. vector (see Section 5.1.15 [deg], page 138)

`det` determinant (see Section 5.1.18 [det], page 140)

`dim` Krull dimension of basering modulo ideal of leading terms, resp. dimension of module of leading terms (see Section 5.1.20 [dim], page 141)

`extgcd` Bezout representation of gcd (see Section 5.1.28 [extgcd], page 146)

`find` position of a substring in a string (see Section 5.1.36 [find], page 151)

`gcd` greatest common divisor (see Section 5.1.41 [gcd], page 154)

`koszul` Koszul matrix (see Section 5.1.64 [koszul], page 170)

`memory` memory usage (see Section 5.1.79 [memory], page 181)

`mult` multiplicity of an ideal, resp. module, of leading terms (see Section 5.1.89 [mult], page 188)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`npars` number of ring parameters (see Section 5.1.93 [npars], page 190)

`nrows` number of rows of a matrix, resp. the rank of the free module where the vector or module lives (see Section 5.1.95 [nrows], page 191)

`nvars` number of ring variables (see Section 5.1.96 [nvars], page 192)

`ord` degree of the leading term of a polynomial resp. vector (see Section 5.1.99 [ord], page 196)

`par` n-th parameter of the basering (see Section 5.1.101 [par], page 197)

`pardeg` degree of a number considered as a polynomial in the ring parameters (see Section 5.1.102 [pardeg], page 197)

`prime` the next lower prime (see Section 5.1.105 [prime], page 199)

`random` a pseudo random integer between the given limits (see Section 5.1.113 [random], page 205)

`regularity`
regularity of a resolution (see Section 5.1.116 [regularity], page 208)

`rvar` test, if the given expression or string is a ring variable (see Section 5.1.122 [rvar], page 213)

`size` number of elements in an object (see Section 5.1.126 [size], page 217)

`trace` trace of an integer matrix (see Section 5.1.139 [trace], page 230)

`var` n-th ring variable of the basering (see Section 5.1.146 [var], page 233)

`vdim` vector space dimension of basering modulo ideal of leading terms, resp. of freemodule modulo module of leading terms (see Section 5.1.149 [vdim], page 234)

4.4.5 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 254.

4.4.6 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
```

4.5 intmat

Integer matrices are matrices with integer entries. For the range of integers see Section 6.1 [Limitations], page 254. 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.

4.5.1 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
```

4.5.2 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 4.5.3 [intmat type cast], page 83)

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 44; Section 4.12 [number], page 108.

4.5.3 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, or intmat. 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.

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
```

See Section 3.5.5 [Type conversion and casting], page 44; Section 4.5 [intmat], page 82; Section 4.10.3 [matrix type cast], page 102.

4.5.4 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
```

4.5.5 intmat related functions

`betti`	Betti numbers of a free resolution (see Section 5.1.3 [betti], page 130)
`det`	determinant (see Section 5.1.18 [det], page 140)
`ncols`	number of cols (see Section 5.1.92 [ncols], page 190)
`nrows`	number of rows (see Section 5.1.95 [nrows], page 191)
`random`	pseudo random intmat (see Section 5.1.113 [random], page 205)
`size`	total number of entries (see Section 5.1.126 [size], page 217)
`transpose`	transpose of an intmat (see Section 5.1.140 [transpose], page 230)
`trace`	trace of an intmat (see Section 5.1.139 [trace], page 230)

4.6 intvec

Variables of type intvec are lists of integers. For the range of integers see Section 6.1 [Limitations], page 254. 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.

4.6.1 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
```

4.6.2 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
```

4.6.3 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)

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
```

4.6.4 intvec related functions

`hilb` Hilbert series as intvec (see Section 5.1.47 [hilb], page 159)

`indepSet` sets of independent variables of an ideal (see Section 5.1.52 [indepSet], page 162)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.68 [leadexp], page 173)

`monomial` the power product corresponding to the exponent vector (see Section 5.1.85 [monomial], page 186)

`nrows` number of rows (see Section 5.1.95 [nrows], page 191)

`qhweight` quasihomogeneous weights (see Section 5.1.110 [qhweight], page 203)

`size` length of the intvec (see Section 5.1.126 [size], page 217)

`sortvec` permutation for sorting ideals/modules (see Section 5.1.128 [sortvec], page 219)

`transpose`
transpose of an intvec, returns an intmat (see Section 5.1.140 [transpose], page 230)

`weight` weights for the weighted ecart method (see Section 5.1.151 [weight], page 235)

4.7 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 4.7.4 [ASCII links], page 89)
- MPfile links (see Section 4.7.5.1 [MPfile links], page 91)
- MPtcp links (see Section 4.7.5.2 [MPtcp links], page 92)
- DBM links (see Section 4.7.6 [DBM links], page 94)

4.7.1 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);
```

4.7.2 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`, `MPfile`, `MPtcp` 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 MPtcp links.

For a detailed format description of the link describing string see:

- for ASCII links: Section 4.7.4 [ASCII links], page 89
- for MPfile links: Section 4.7.5.1 [MPfile links], page 91
- for MPtcp links: Section 4.7.5.2 [MPtcp links], page 92
- for DBM links: Section 4.7.6 [DBM links], page 94

4.7.3 link related functions

`close`	closes a link (see Section 5.1.9 [close], page 133)
`dump`	generates a dump of all variables and their values (see Section 5.1.22 [dump], page 142)
`getdump`	reads a dump (see Section 5.1.43 [getdump], page 155)
`open`	opens a link (see Section 5.1.97 [open], page 192)
`read`	reads from a link (see Section 5.1.114 [read], page 206)
`status`	gets the status of a link (see Section 5.1.132 [status], page 222)
`write`	writes to a link (see Section 5.1.153 [write], page 236)
`kill`	closes and kills a link (see Section 5.1.62 [kill], page 170)

4.7.4 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 250). 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 5.1.27 [execute], page 145).

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 **
  close(l);                    // link is closed
```

4.7.5 MP links

MP (Multi Protocol) links give the possibility to store and communicate data in the binary MP format: Read and write access is very fast compared to ASCII links. MP links can be established using files (link type is `MPfile`) or using TCP sockets (link type is `MPtcp`). All data (including such data that cannot be converted to a string) can be written to an MP link. For ring-dependent data, a ring description is written together with the data. Reading from an MP link returns an

expression (not a string) which was evaluated after the read operation. If the expression read from an MP link is not from the same ring as the current ring, then a `read` changes the current ring.

Note: Currently, MP links are only available on Unix platforms and data is written without attributes (which is likely to change in future versions). For a general description of MP, see `http://www.symbolicnet.org/areas/protocols/mp.html`.

4.7.5.1 MPfile links

MPfile links provide the possibility to store data in a file using the binary MP format. Read and write operations are very fast compared to ASCII links. Therefore, for storing large amounts of data, MPfile links should be used instead of ASCII links. Unlike ASCII links, data read from MPfile links is returned as expressions one at a time, and not as a string containing the entire content of the file. Furthermore, ring-dependent data is stored together with a ring description. Therefore, reading ring-dependent data might change the current ring.

The MPfile link describing string has to be one of the following:

1. `"MPfile: " +` filename
   the mode (read or append) is set by the first `read` or `write` command.
2. `"MPfile:r " +` filename
   opens the file for reading.
3. `"MPfile:w " +` filename
   opens the file for overwriting.
4. `"MPfile:a " +` filename
   opens the file for appending.

There are the following default values:

- if none 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).

Note that the filename may contain a path. An MPfile 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="MPfile:w example.mp"; // type=MPfile, mode=overwrite
  l;
↦ // type : MPfile
↦ // mode : w
↦ // name : example.mp
↦ // open : no
↦ // read : not ready
↦ // write: not ready
  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);
↦ // mpsr_r0              [0]  *ring
  def s = read(l);             // third expression
  listvar();
↦ // s                   [0]  string hello world
↦ // mpsr_r0              [0]  *ring
↦ //      i                   [0]  ideal, 3 generator(s)
↦ // l                   [0]  link
  close(l);                    // link is closed
  dump("MPfile:w example.mp"); // dump everything to example.mp
  kill i, s;                   // kill i and s
  getdump("MPfile: example.mp");// get previous dump
  listvar();                   // got all variables and values back
↦ // mpsr_r0              [0]  *ring
↦ //      i                   [0]  ideal, 3 generator(s)
↦ // s                   [0]  string hello world
↦ // l                   [0]  link
```

4.7.5.2 MPtcp links

MPtcp links give the possibility to exchange data in the binary MP format between two processes which may run on the same or on different computers. MPtcp links can be opened in four different modes:

`listen` SINGULAR acts as a server.

`connect` SINGULAR acts as a client.

`launch` SINGULAR acts as a client, launching an application as server.

`fork` SINGULAR acts as a client, forking another SINGULAR as server.

The MPtcp link describing string has to be

- listen mode:
  1. `"MPtcp:listen --MPport " +` portnumber

  SINGULAR becomes a server and waits at the port for a connect call.
- connect mode:
  2. `"MPtcp:connect --MPport " +` portnumber
  3. `"MPtcp:connect --MPhost " +` hostname `+ " --MPport " +` portnumber

  SINGULAR becomes a client and connects to a server waiting at the host and port.
- launch mode:
  4. `"MPtcp:launch"`
  5. `"MPtcp:launch --MPrsh " +` rsh
  6. `"MPtcp:launch --MPrsh " +` rsh `+ " --MPhost " +` hostname
  7. `"MPtcp:launch --MPrsh " +` rsh `+ " --MPhost " +` hostname `+ " --MPapplication " +` application

  SINGULAR becomes a client and starts (launches) the application using the specified remote shell command (default is `ssh`) on a (possibly) different host (default is `localhost` which then acts as a server.

- fork mode:

  8. `"MPtcp:fork"`

  SINGULAR becomes a client and forks another SINGULAR on the same host which acts as a server.

There are the following default values:

- if none of `listen`, `connect`, `launch` or `fork` is specified, the default mode is set to `fork`.
- if no remote shell (rsh) command is specified, then the command `ssh` is used.
- if no application is specified in mode `launch` the default application is the value of `system("Singular") + "-bq"`. (This evaluates to the absolute path of the SINGULAR currently running with the option `"-bq"` appended.)
- if no hostname is specified the local host is used as default host.

To open an MPtcp 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. SINGULAR automatically appends the command line arguments `"--MPmode connect --MPhost` hostname `--MPport` portnumber`"` to the command line of the server application. Both hostname and portnumber are substituted by the values from the link specification. The client "listens" at the given port until the server application does a connect call. If SINGULAR is used as server application it has to be started with the command line option `-b`. Since launching is done using a remote shell command, the host on which the application should run must have an entry in the `.rhosts` file. Even the local machine must have an entry if applications are to be launched locally.

If the MPtcp 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 5.1.111 [quote], page 204), 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 `mp_ll` is the MPtcp 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 `mp_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.

MPtcp links should explicitly be opened before being used. MPtcp links are bidirectional, i.e. can be used for both, writing and reading. Reading from an MPtcp link blocks until data was written to that link. The `status` command can be used to check whether there is data to read.

Example:

```
LIB "general.lib"; // needed for "killall" command
link l="MPtcp:launch";
open(l); l;        // l is ready for writing but not for reading
↦ // type : MPtcp
↦ // mode : launch
↦ // name :
↦ // open : yes
↦ // read : not ready
```

```
↦ // write: ready

  ring r; ideal i=x2+y,xyz+z,x2+y2;

  write (l,quote(std(eval(i))));   // std(i) is computed on server
  def j = read(l);j; // result of computation on server is read
↦ j[1]=z
↦ j[2]=y2-y
↦ j[3]=x2+y2

  write(l, quote(getdump(mp_ll))); // server reads dump
  dump(l);           //  dump is written to server (includes proc's)
  read(l);           // result of previous write-command is read
  killall("not", "link"); killall("proc"); // kills everything, but links

  write(l, quote(dump(mp_ll)));    // server writes dump
  getdump(l);        // dump is read from server
  read(l);           // result of previous write-command is read

  close(l);          // server is shut down
  listvar(all);      // same state as we had before "killall()"
↦ // mpsr_r0              [0]  ring
↦ // r                   [0]  *ring
↦ //      j                   [0]  ideal, 3 generator(s)
↦ //      i                   [0]  ideal, 3 generator(s)
↦ // l                   [0]  link

  l = "MPtcp:";      // fork link declaration
  open(l); l;        // Notice that name is "parent"
↦ // type : MPtcp
↦ // mode : fork
↦ // name : parent
↦ // open : yes
↦ // read : not ready
↦ // write: ready

  write(l, quote(status(mp_ll, "name")));
  read(l);           // and name of forked link is "child"
↦ child
  write(l,quote(i)); // Child inherited vars and their values
  read(l);
↦ _[1]=x2+y
↦ _[2]=xyz+z
↦ _[3]=x2+y2
  close(l);          // shut down forked child
```

4.7.6 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);
```

4.8 list

Lists are arrays whose elements can be of different types (including ring and qring). 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 and qrings 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.

4.8.1 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)
```

4.8.2 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 44.

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
```

4.8.3 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
```

4.8.4 list related functions

`bareiss` returns a list of a matrix (lower triangular) and of an intvec (permutations of columns, see Section 5.1.2 [bareiss], page 128)

`betti` Betti numbers of a resolution (see Section 5.1.3 [betti], page 130)

`delete` deletion of an element from a list (see Section 5.1.17 [delete], page 139)

`facstd` factorizing Groebner basis algorithm (see Section 5.1.29 [facstd], page 146)

`factorize` list of factors of a polynomial (see Section 5.1.30 [factorize], page 147)

`insert` insertion of a new element into a list (see Section 5.1.53 [insert], page 163)

`lres` free resolution (see Section 5.1.74 [lres], page 177)

`minres` minimization of a free resolution (see Section 5.1.82 [minres], page 184)

`mres` minimal free resolution of an ideal, resp. module w.r.t. a minimal set of generators of the first module (see Section 5.1.87 [mres], page 186)

`names` list of all user-defined variable names (see Section 5.1.91 [names], page 189)

`res` free resolution of an ideal, resp. module (see Section 5.1.118 [res], page 209)

`size` number of entries (see Section 5.1.126 [size], page 217)

`sres` free resolution of an ideal, resp. module, given by a standard base (see Section 5.1.131 [sres], page 221)

4.9 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 260, Section 3.7.2 [Names in procedures], page 51.

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 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.

4.9.1 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 4.9 [map], page 98.

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 5.1.32 [fetch], page 148; Section 4.3.2 [ideal expressions], page 73; Section 5.1.50 [imap], page 161; Section 4.9 [map], page 98; Section 4.18 [ring], page 118.

4.9.2 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 (`(`, `)`)

4.9.3 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
```

4.9.4 map related functions

`fetch` the identity map between rings and qrings (see Section 5.1.32 [fetch], page 148)

`imap` a convenient map procedure for inclusions and projections of rings (see Section 5.1.50 [imap], page 161)

`preimage` preimage under a ring map (see Section 5.1.104 [preimage], page 198)

`subst` substitution of a ring variable (see Section 5.1.136 [subst], page 226)

See also the libraries Section D.4.2 [algebra_lib], page 653 and Section D.2.8 [ring_lib], page 604, which contain more functions, related to maps.

4.10 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.

4.10.1 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
```

4.10.2 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 4.10.3 [matrix type cast], page 102)

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
```

4.10.3 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 44; Section 4.5.3 [intmat type cast], page 83; Section 4.10 [matrix], page 101.

4.10.4 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
```

4.10.5 matrix related functions

`bareiss` Gauss-Bareiss algorithm (see Section 5.1.2 [bareiss], page 128)

`coef` matrix of coefficients and monomials (see Section 5.1.10 [coef], page 134)

`coeffs` matrix of coefficients (see Section 5.1.11 [coeffs], page 134)

`det` determinant (see Section 5.1.18 [det], page 140)

`diff` partial derivative (see Section 5.1.19 [diff], page 140)

`jacob` Jacobi matrix (see Section 5.1.57 [jacob], page 166)

`koszul` Koszul matrix (see Section 5.1.64 [koszul], page 170)

`lift` lift-matrix (see Section 5.1.71 [lift], page 174)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.72 [liftstd], page 175)

`minor` set of minors of a matrix (see Section 5.1.81 [minor], page 182)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`nrows` number of rows (see Section 5.1.95 [nrows], page 191)

`print` nice print format (see Section 5.1.107 [print], page 200)

`size` number of matrix entries (see Section 5.1.126 [size], page 217)

`subst` substitute a ring variable (see Section 5.1.136 [subst], page 226)

`trace` trace of a matrix (see Section 5.1.139 [trace], page 230)

`transpose`
transposed matrix (see Section 5.1.140 [transpose], page 230)

`wedge` wedge product (see Section 5.1.150 [wedge], page 235)

See also the library Section D.3.1 [matrix_lib], page 614, which contains more matrix-related functions.

4.11 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 501).

4.11.1 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
```

4.11.2 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 44; Section 4.3 [ideal], page 72; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

4.11.3 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
```

4.11.4 module related functions

`coeffs` matrix of coefficients (see Section 5.1.11 [coeffs], page 134)

`degree` multiplicity, dimension and codimension of the module of leading terms (see Section 5.1.16 [degree], page 139)

`diff` partial derivative (see Section 5.1.19 [diff], page 140)

`dim` Krull dimension of free module over the basering modulo the module of leading terms (see Section 5.1.20 [dim], page 141)

`eliminate`
elimination of variables (see Section 5.1.23 [eliminate], page 143)

`freemodule`
the free module of given rank (see Section 5.1.39 [freemodule], page 153)

`groebner` Groebner basis computation (a wrapper around `std,stdhilb,stdfglm`,...) (see Section 5.1.44 [groebner], page 155)

`hilb` Hilbert function of a standard basis (see Section 5.1.47 [hilb], page 159)

`homog` homogenization with respect to a variable (see Section 5.1.48 [homog], page 160)

`interred` interreduction of a module (see Section 5.1.55 [interred], page 165)

`intersect` module intersection (see Section 5.1.56 [intersect], page 165)

`jet` Taylor series up to a given order (see Section 5.1.59 [jet], page 167)

`kbase` vector space basis of free module over the basering modulo the module of leading terms (see Section 5.1.60 [kbase], page 168)

`lead` initial module (see Section 5.1.66 [lead], page 172)

`lift` lift-matrix (see Section 5.1.71 [lift], page 174)

`liftstd` standard basis and transformation matrix computation (see Section 5.1.72 [liftstd], page 175)

`lres` free resolution (see Section 5.1.74 [lres], page 177)

`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 5.1.83 [modulo], page 185)

`mres` minimal free resolution of an ideal resp. module w.r.t. a minimal set of generators of the given module (see Section 5.1.87 [mres], page 186)

`mult` multiplicity, resp. degree, of the module of leading terms (see Section 5.1.89 [mult], page 188)

`nres` computation of a free resolution of an ideal resp. module M which is minimized from the second free module on (see Section 5.1.94 [nres], page 190)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`nrows` number of rows (see Section 5.1.95 [nrows], page 191)

`print` nice print format (see Section 5.1.107 [print], page 200)

`prune` minimization of the embedding into a free module (see Section 5.1.109 [prune], page 203)

`qhweight` quasihomogeneous weights of an ideal, resp. module (see Section 5.1.110 [qhweight], page 203)

`quotient` module quotient (see Section 5.1.112 [quotient], page 204)

`reduce` normalform with respect to a standard basis (see Section 5.1.115 [reduce], page 206)

`res` free resolution of an ideal, resp. module, but not changing the given ideal, resp. module (see Section 5.1.118 [res], page 209)

`simplify` simplification of a set of vectors (see Section 5.1.125 [simplify], page 216)

`size` number of non-zero generators (see Section 5.1.126 [size], page 217)

`sortvec` permutation for sorting ideals/modules (see Section 5.1.128 [sortvec], page 219)

`sres` free resolution of a standard basis (see Section 5.1.131 [sres], page 221)

`std` standard basis computation (see Section 5.1.133 [std], page 223, Section 5.1.72 [liftstd], page 175)

`subst` substitution of a ring variable (see Section 5.1.136 [subst], page 226)

`syz` computation of the first syzygy module (see Section 5.1.138 [syz], page 229)

`vdim` vector space dimension of free module over the basering modulo module of leading terms (see Section 5.1.149 [vdim], page 234)

`weight` "optimal" weights (see Section 5.1.151 [weight], page 235)

4.12 number

Numbers are elements from the coefficient field (or ground field). They can only be defined or accessed with respect to a basering which determines the coefficient field. See Section 4.18.1 [ring declarations], page 119 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 258.

4.12.1 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 29 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)
```

4.12.2 number expressions

A number expression is:

1. a rational number (there are NO spaces allowed inside a rational number, see Section 4.4.2 [int expressions], page 77)
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 44)
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;
```

```
↦ 0
  4/ 8;
↦ 0
  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 44; Section 4.18 [ring], page 118.

4.12.3 number operations

`+` addition

`-` negation or subtraction

`*` multiplication

`/` division

`^`, `**` 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 258.
For the behavior of comparison operators in rings with ground field different from real or the rational numbers, see Section 4.4.5 [boolean expressions], page 80.

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);
↦ 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
```

4.12.4 number related functions

`cleardenom`
: cancellation of denominators of numbers in polyomial and divide it by its content (see Section 5.1.8 [cleardenom], page 133)

`impart`
: imaginary part of a complex number, 0 otherwise (see Section 5.1.51 [impart], page 162, Section 5.1.117 [repart], page 208)

`numerator, denominator`
: the numerator/denominator of a rational number (see Section D.2.5.15 [numerator], page 557, Section D.2.5.16 [denominator], page 557)

`leadcoef`
: coefficient of the leading term (see Section 5.1.67 [leadcoef], page 172)

`par`
: n-th parameter of the basering (see Section 5.1.101 [par], page 197)

`pardeg`
: degree of a number in ring parameters (see Section 5.1.102 [pardeg], page 197)

`parstr`
: string form of ring parameters (see Section 5.1.103 [parstr], page 198)

`repart`
: real part of a complex number (see Section 5.1.51 [impart], page 162, Section 5.1.117 [repart], page 208)

4.13 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 53.

4.13.1 package declarations

Syntax: `package` name `;`

Purpose: defines a package (Only relevant in very special situations).

Example:

```
  package Test;
  int Test::i;
  listvar();
  listvar(Test);
↦ // Test                  [0]  package (N)
↦ // ::i                   [0]  int 0
  package dummy = Test;
  kill Test;
  listvar(dummy);
↦ // dummy                 [0]  package (N)
↦ // ::i                   [0]  int 0
```

4.13.2 package related functions

`exportto` transfer an identifier to the specified package (see Section 5.2.6 [exportto], page 240)

`importfrom`
generate a copy of an identifier from the specified package in the current package (see Section 5.2.9 [importfrom], page 242)

`listvar` list variables currently defined in a given package (see Section 5.1.73 [listvar], page 176)

`load` load a library or dynamic module (see Section 5.2.11 [load], page 245)

`LIB` load a library or dynamic module (see Section 5.1.70 [LIB], page 174)

4.14 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 4.14.2 [poly expressions], page 113). 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;
```

4.14.1 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 250

4.14.2 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 44)
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 44; Section 4.18 [ring], page 118.

4.14.3 poly operations

`+` addition

`-` negation or subtraction

`*` multiplication

`/` division by a polynomial, non divisible terms yield 0

`^`, `**` 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
```

4.14.4 poly related functions

`cleardenom`
: cancellation of denominators of numbers in polynomial and divide it by its content (see Section 5.1.8 [cleardenom], page 133; Section D.2.5.14 [content], page 557)

`coef` matrix of coefficients and monomials (see Section 5.1.10 [coef], page 134)

`coeffs` matrix of coefficients (see Section 5.1.11 [coeffs], page 134)

`deg` degree (see Section 5.1.15 [deg], page 138)

`diff` partial derivative (see Section 5.1.19 [diff], page 140)

`extgcd` Bezout representation of gcd (see Section 5.1.28 [extgcd], page 146)

`factorize`
: factorization of polynomial (see Section 5.1.30 [factorize], page 147)

`finduni` univariate polynomials in a zero-dimensional ideal (see Section 5.1.37 [finduni], page 151)

`gcd` greatest common divisor (see Section 5.1.41 [gcd], page 154)

`homog` homogenization (see Section 5.1.48 [homog], page 160)

`jacob` ideal, resp. matrix, of all partial derivatives (see Section 5.1.57 [jacob], page 166)

`lead` leading term (see Section 5.1.66 [lead], page 172)

`leadcoef` coefficient of the leading term (see Section 5.1.67 [leadcoef], page 172)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.68 [leadexp], page 173)

`leadmonom`
: leading monomial (see Section 5.1.69 [leadmonom], page 173)

`jet` monomials of degree at most k (see Section 5.1.59 [jet], page 167)

`ord` degree of the leading monomial (see Section 5.1.99 [ord], page 196)

`qhweight` quasihomogeneous weights (see Section 5.1.110 [qhweight], page 203)

`reduce` normal form with respect to a standard base (see Section 5.1.115 [reduce], page 206)

`rvar` test for ring variable (see Section 5.1.122 [rvar], page 213)

`simplify` normalization of a polynomial (see Section 5.1.125 [simplify], page 216)

`size` number of monomials (see Section 5.1.126 [size], page 217)

`subst` substitution of a ring variable (see Section 5.1.136 [subst], page 226)

`trace` trace of a matrix (see Section 5.1.139 [trace], page 230)

`var` the indicated variable of the ring (see Section 5.1.146 [var], page 233)

`varstr` variable(s) in string form (see Section 5.1.148 [varstr], page 234)

4.15 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 49. 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 53.

4.15.1 proc declaration

Syntax:

```
[static] proc proc_name [parameter_list]
["help_text"]
{
  procedure_body
}
[example
{
  sequence_of_commands;
}]
proc proc_name = proc_name ;
proc proc_name = string_expression ;
```

Purpose: defines a new function, the `proc` proc_name, with the additional information help_text, which is copied to the screen by `help proc_name;` and the `example` section which is executed by `example proc_name;`.
The help_text, the parameter_list, and the example section are optional. The default for a parameter_list is `(list #)`, see Section 3.7.3 [Parameter list], page 51. The help and example sections are ignored if the procedure is defined interactively, i.e., if it was not loaded from a file by a `LIB` command (see Section 5.1.70 [LIB], page 174).
Specifying `static` in front of the proc-definition is only possible in a library file and makes this procedure local to the library, i.e., accessible only for the other procedures in the same library, but not for the users.

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 5.1.70 [LIB], page 174; Section 3.8 [Libraries], page 53.

4.16 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 29.

4.16.1 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.

Example:

```
ring r=0,(x,y,z),dp;
ideal i=xy;
qring q=std(i);
basering;
↦ //   characteristic : 0
↦ //   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
```

4.17 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 508).

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).

4.17.1 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
↦ resolution not minimized yet
↦
  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
```

4.17.2 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 44.

4.17.3 resolution related functions

`betti`	Betti numbers of a resolution (see Section 5.1.3 [betti], page 130)
`lres`	free resolution (see Section 5.1.74 [lres], page 177)
`minres`	minimize a free resolution (see Section 5.1.82 [minres], page 184)
`mres`	minimal free resolution of an ideal, resp. module and a minimal set of generators of the given ideal, resp. module (see Section 5.1.87 [mres], page 186)
`res`	free resolution of an ideal, resp. module, but not changing the given ideal, resp. module (see Section 5.1.118 [res], page 209)
`sres`	free resolution of a standard basis (see Section 5.1.131 [sres], page 221)

4.18 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 29.

4.18.1 ring declarations

Syntax: `ring` name `=` `(` coefficients `),` `(` names_of_ring_variables `),` `(` ordering `);`

Default: `32003,(x,y,z),(dp,C);`

Purpose: declares a ring and sets it as the actual basering.

The coefficients are given by one of the following:

1. a non-negative int_expression less or equal 2147483629.
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.
6. an expression_list of the name `integer`.
7. an expression_list of the name `integer` and following int_expressions.
8. 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 29.

'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`, 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 3.3.3 [Term orderings], page 33, Section B.2 [Monomial orderings], page 502.

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 30; Section 4.18 [ring], page 118; Section 5.1.121 [ringlist], page 211.

4.18.2 ring related functions

`charstr`	description of the coefficient field of a ring (see Section 5.1.6 [charstr], page 132)
`keepring`	move ring to next upper level (see Section 5.2.10 [keepring], page 244)
`npars`	number of ring parameters (see Section 5.1.93 [npars], page 190)
`nvars`	number of ring variables (see Section 5.1.96 [nvars], page 192)
`ordstr`	monomial ordering of a ring (see Section 5.1.100 [ordstr], page 197)
`parstr`	names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.103 [parstr], page 198)
`qring`	quotient ring (see Section 4.16 [qring], page 117)
`ringlist`	decomposition of a ring into a list of its components (see Section 5.1.121 [ringlist], page 211)
`setring`	setting of a new basering (see Section 5.1.123 [setring], page 213)
`varstr`	names of all ring variables or the name of the n-th ring variable (see Section 5.1.148 [varstr], page 234)

4.18.3 ring operations

+ construct a new ring $k[X,Y]$ from $k_1[X]$ and $k_2[Y]$. (The sets of variables must be distinct).

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;
↦ //   characteristic : 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.2.8 [ring_lib], page 604

4.19 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 5.1.27 [execute], page 145. They are also return values of certain interpreter related functions (see Section 5.1 [Functions], page 127). 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.

4.19.1 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...
```

4.19.2 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 4.19.3 [string type cast], page 122)
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 44

4.19.3 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);
↦ (32003),(x,y,z),(dp(3),C)
  string(ideal(x,y));
↦ x,y
  qring R = std(ideal(x,y));
  string(R);
↦ (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 44; Section 5.1.107 [print], page 200; Section 4.19 [string], page 120.

4.19.4 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
```

4.19.5 string related functions

`charstr` description of the coefficient field of a ring (see Section 5.1.6 [charstr], page 132)

`execute` executing string as command (see Section 5.1.27 [execute], page 145)

`find` position of a substring in a string (see Section 5.1.36 [find], page 151)

`names` list of strings of all user-defined variable names (see Section 5.1.91 [names], page 189)

`nameof` name of an object (see Section 5.1.90 [nameof], page 188)

`option` lists all defined options (see Section 5.1.98 [option], page 192)

`ordstr` monomial ordering of a ring (see Section 5.1.100 [ordstr], page 197)

`parstr` names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.103 [parstr], page 198)

`read` read a file (see Section 5.1.114 [read], page 206)

`size` length of a string (see Section 5.1.126 [size], page 217)

`sprintf` string formatting (see Section 5.1.130 [sprintf], page 220)

`typeof` type of an object (see Section 5.1.142 [typeof], page 231)

`varstr` names of all ring variables or the name of the n-th ring variable (see Section 5.1.148 [varstr], page 234)

4.20 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 501).

4.20.1 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]
```

4.20.2 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 44; Section 4.18 [ring], page 118.

4.20.3 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
```

4.20.4 vector related functions

`cleardenom`
quotient of a vector by its content (see Section 5.1.8 [cleardenom], page 133)

`coeffs` matrix of coefficients (see Section 5.1.11 [coeffs], page 134)

`deg` degree (see Section 5.1.15 [deg], page 138)

`diff` partial derivative (see Section 5.1.19 [diff], page 140)

`gen` i-th generator (see Section 5.1.42 [gen], page 154)

`homog` homogenization (see Section 5.1.48 [homog], page 160)

`jet` k-jet: monomials of degree at most k (see Section 5.1.59 [jet], page 167)

`lead` leading term (see Section 5.1.66 [lead], page 172)

`leadcoef` leading coefficient (see Section 5.1.67 [leadcoef], page 172)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.68 [leadexp], page 173)

`leadmonom`
leading monomial (see Section 5.1.69 [leadmonom], page 173)

`nrows` number of rows (see Section 5.1.95 [nrows], page 191)

`ord` degree of the leading monomial (see Section 5.1.99 [ord], page 196)

`reduce` normal form with respect to a standard base (see Section 5.1.115 [reduce], page 206)

`simplify` normalize a vector (see Section 5.1.125 [simplify], page 216)

`size` number of monomials (see Section 5.1.126 [size], page 217)

`subst` substitute a ring variable (see Section 5.1.136 [subst], page 226)

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 525, 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 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
// although maxideal(2) is a standard basis,
// SINGULAR does not know it:
attrib(maxideal(2), "isSB");
↦ 0
```

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);
```

```
↦ // ** I is no standard basis
↦ 4
  attrib(I,"isSB",1);  // the standard basis attribute is set here
  vdim(I);
↦ 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:
(isSB, isHomog are used by the kernel, the other are used by libraries)

`isSB`	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`	the weight vector for homogeneous or quasihomogeneous ideals/modules
`isCI`	complete intersection property
`isCM`	Cohen-Macaulay property
`rank`	set the rank of a module (see Section 5.1.95 [nrows], page 191)
`withSB`	value of type ideal, resp. module, is std
`withHilb`	value of type intvec is hilb(_,1) (see Section 5.1.47 [hilb], page 159)
`withRes`	value of type list is a free resolution
`withDim`	value of type int is the dimension (see Section 5.1.20 [dim], page 141)
`withMult`	value of type int is the multiplicity (see Section 5.1.89 [mult], page 188)

5.1.2 bareiss

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 523, 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 5.1.18 [det], page 140; Section 4.10 [matrix], page 101.

5.1.3 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/sres/mres/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.

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 508; Section 5.1.49 [hres], page 161; Section 5.1.74 [lres], page 177; Section 5.1.87 [mres], page 186; Section 5.1.107 [print], page 200; Section 5.1.118 [res], page 209; Section 4.17 [resolution], page 117; Section 5.1.131 [sres], page 221.

5.1.4 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 5.1.6 [charstr], page 132; Section 4.18 [ring], page 118.

5.1.5 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 509.

5.1.6 charstr

Syntax: `charstr` (ring_name)

Type: string

Purpose: returns the description of the coefficient field of a ring.

Example:

```
  ring r=32003,(x,y),dp;
  charstr(r);
↦ 32003
  ring s=0,(x,y),dp;
  charstr(s);
↦ 0
  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);
↦ real
```

See Section 5.1.4 [char], page 131; Section 5.1.100 [ordstr], page 197; Section 4.18 [ring], page 118; Section 5.1.148 [varstr], page 234.

5.1.7 chinrem

Syntax: `chinrem` (list, intvec)
`chinrem` (list, list)
`chinrem` (intvec, intvec)

Type: ideal resp. bigint

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 `ideal`, `module` or `matrix`. The moduli, if given by a list, must be of type `bigint` or `int`.

Example:

```
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(intvec(2,-3),intvec(7,11));
↦ 30
```

See Section D.4.10 [modstd_lib], page 710.

5.1.8 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.2.5.14 [content], page 557.

5.1.9 close

Syntax: `close ( link_expression )`

Type: none

Purpose: closes a link.

Example:

```
link l="MPtcp:launch";
open(l); // start SINGULAR "server" on localhost in batchmode
close(l); // shut down SINGULAR server
```

See Section 4.7 [link], page 88; Section 5.1.97 [open], page 192.

5.1.10 coef

Syntax: `coef ( poly_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: 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 5.1.11 [coeffs], page 134) 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
  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 5.1.11 [coeffs], page 134.

5.1.11 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

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 occuring 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
```

See Section 5.1.10 [coef], page 134; Section 5.1.60 [kbase], page 168.

5.1.12 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 5.1.19 [diff], page 140.

5.1.13 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 3.9 [Guidelines for writing a library], 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 3.10 [Debugging tools], page 66; Section 5.1.107 [print], page 200; Section 5.3.6 [printlevel], page 249; Section 5.3.11 [voice], page 253.

5.1.14 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 5.1.122 [rvar], page 213; Section 5.3.11 [voice], page 253.

5.1.15 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 5.1.59 [jet], page 167; Section 5.1.99 [ord], page 196; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.16 degree

Syntax: `degree ( ideal_expression )`
`degree ( module_expression )`

Type: none

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 5.1.20 [dim], page 141; Section 4.3 [ideal], page 72; Section 5.1.89 [mult], page 188; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.17 delete

Syntax: `delete ( list_expression, int_expression )`

Type: list

Purpose: deletes the element with the given index from a list (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
```

See Section 5.1.53 [insert], page 163; Section 4.8 [list], page 96.

5.1.18 det

Syntax: `det ( intmat_expression )`
`det ( matrix_expression )`
`det ( module_expression )`

Type: int, resp. poly

Purpose: returns the determinant of a square matrix. A module is considered as a matrix. The applied algorithms depend on type of input. If the input is a module or matrix with symbolic entries the Bareiss algorithm is used. In the other cases the chinese remainder algorithm is used. For large sparse problems the input as a module has advantages.

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 4.5 [intmat], page 82; Section 4.10 [matrix], page 101; Section 5.1.81 [minor], page 182.

5.1.19 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

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.

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
```

See Section 5.1.12 [contract], page 136; Section 4.3 [ideal], page 72; Section 5.1.57 [jacob], page 166; Section 4.10 [matrix], page 101; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 5.1.146 [var], page 233; Section 4.20 [vector], page 124.

5.1.20 dim

Syntax: `dim` (ideal_expression)
`dim` (module_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.

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.

Example:

```
  ring r=32003,(x,y,z),dp;
  ideal I=x2-y,x3;
  dim(std(I));
↦ 1
```

See Section 5.1.16 [degree], page 139; Section 4.3 [ideal], page 72; Section 5.1.89 [mult], page 188; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.21 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 4.3 [ideal], page 72; Section 5.1.71 [lift], page 174; Section 4.11 [module], page 105.

5.1.22 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 MP 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");
↦ // ** redefining Standard **
↦ // ** redefining stdfglm **
↦ // ** redefining stdhilb **
↦ // ** redefining quotientList **
↦ // ** redefining par2varRing **
↦ // ** redefining hilbRing **
↦ // ** redefining qslimgb **
↦ // ** redefining groebner **
↦ // ** redefining res **
↦ // ** redefining quot **
↦ // ** redefining quotient1 **
↦ // ** redefining quotient2 **
↦ // ** redefining quotient3 **
↦ // ** redefining quotient5 **
↦ // ** redefining quotient4 **
↦ // ** redefining sprintf **
↦ // ** redefining printf **
↦ // ** redefining fprintf **
↦ // ** redefining weightKB **
r;
↦ //   characteristic : 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 5.1.43 [getdump], page 155; Section 4.7 [link], page 88; Section 5.1.153 [write], page 236.

5.1.23 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, intvec_hilb)
`eliminate` (module_expression, product_of_ring_variables, intvec_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 5.1.47 [hilb], page 159; Section 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223.

5.1.24 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 MPfile link, with `quote` and `write` to prevent local evaluations when writing to an MPtcp link.

Example:

```
link l="MPfile:w example.mp";
ring r=0,(x,y,z),ds;
ideal i=maxideal(3);
ideal j=x7,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);
↦ [1023:1]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 4.7.5.1 [MPfile links], page 91; Section 5.1.111 [quote], page 204; Section 5.1.153 [write], page 236.

5.1.25 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
i;
↦ 1
```

5.1.26 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 5.1.45 [help], page 157

5.1.27 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
```

5.1.28 extgcd

Syntax: `extgcd ( int_expression, int_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]:
↦    2x5+2x4
↦ [2]:
↦    x2+x+1
↦ [3]:
↦    1
```

See Section 5.1.41 [gcd], page 154; Section 4.4 [int], page 77.

5.1.29 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 5.1.30 [factorize], page 147 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 4.3 [ideal], page 72; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223.

5.1.30 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 and finite fields of type `(p^n,a)`.

Example:

```
  ring r=32003,(x,y,z),dp;
  factorize(9*(x-1)^2*(y+z));
↦ [1]:
↦    _[1]=9
↦    _[2]=x-1
↦    _[3]=y+z
↦ [2]:
↦    1,2,1
  factorize(9*(x-1)^2*(y+z),1);
↦ _[1]=x-1
↦ _[2]=y+z
  factorize(9*(x-1)^2*(y+z),2);
↦ [1]:
↦    _[1]=x-1
↦    _[2]=y+z
```

```
↦ [2]:
↦    2,1
  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 652; Section 4.14 [poly], page 112.

5.1.31 farey

Syntax: `farey ( bigint_expression , bigint_expression )`
`farey ( ideal_expression , bigint_expression )`
`farey ( module_expression , bigint_expression )`
`farey ( matrix_expression , bigint_expression )`

Type: type of the first argument

Purpose: lift the first argument modulo the second to the rationals

Note: The current coefficient field must be the rationals.

Example:

```
  ring r=0,x,dp;
  farey(2,32003);
↦ 2
```

See Section 5.1.7 [chinrem], page 132.

5.1.32 fetch

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 4.9 [map], page 98 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:

```
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
```

See Section 5.1.50 [imap], page 161; Section 4.9 [map], page 98; Section 4.16 [qring], page 117; Section 4.18 [ring], page 118.

5.1.33 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 5.1.34 [fglmquot], page 150; Section 5.1.98 [option], page 192; Section 4.16 [qring], page 117; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section 5.1.149 [vdim], page 234.

5.1.34 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 5.1.33 [fglm], page 149; Section 5.1.98 [option], page 192; Section 5.1.112 [quotient], page 204; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.35 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 5.1.27 [execute], page 145; Section 5.1.114 [read], page 206.

5.1.36 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 4.19 [string], page 120.

5.1.37 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 5.1.98 [option], page 192; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.38 fprintf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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:(0),(x,y,z),(dp(3),C)
fprintf(li,"s:%s",list(l));
↦ s:(0),(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("(0),(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]:
↦    //   characteristic : 0
↦ //   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]:
↦    //   characteristic : 0
↦ //   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 5.1.107 [print], page 200; Section 5.1.108 [printf], page 202; Section 5.1.130 [sprintf], page 220; Section 4.19 [string], page 120.

5.1.39 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 5.1.42 [gen], page 154; Section 4.11 [module], page 105.

5.1.40 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 5.1.33 [fglm], page 149; Section 5.1.44 [groebner], page 155; Section 4.16 [qring], page 117; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223.

5.1.41 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 4.1 [bigint], page 70; Section 5.1.28 [extgcd], page 146; Section 4.4 [int], page 77; Section 4.12 [number], page 108.

5.1.42 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 5.1.39 [freemodule], page 153; Section 4.4 [int], page 77; Section 4.20 [vector], page 124.

5.1.43 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 MP 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 5.1.22 [dump], page 142; Section 4.7 [link], page 88; Section 5.1.114 [read], page 206.

5.1.44 groebner

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

Syntax:
`groebner ( ideal_expression )`
`groebner ( module_expression )`
`groebner ( ideal_expression, int_expression )`
`groebner ( module_expression, int_expression )`
`groebner ( ideal_expression, list of string_expressions )`
`groebner ( ideal_expression, list of string_expressions and int_expression )`
`groebner ( ideal_expression, 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).
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.

Note: If an additional argument, say wait, of type int is given, then the computation runs for at most wait seconds. That is, if no result could be computed in wait seconds, then the computation is interrupted, 0 is returned, a warning message is displayed, and the global variable Standard::groebner_error is defined. This feature uses MP and hence it is available on UNIX platforms, only.

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 (0),(a,b,c,d),(dp(4),C)
↦ [255:2]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 (0),(a,b,c,d,@),(dp(5),C)
↦ weights used for hilbert series: 1,1,1,1,1
↦ [63:1]1(3)s2(2)s3s4-s5ss6-s7--
↦ product criterion:8 chain criterion:5
↦ std with hilb in (0),(a,b,c,d,@),(lp(4),dp(1),C)
↦ [63:1]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 (0),(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 (0),(b,c,d,a,@),(dp(5),C)
↦ weights used for hilbert series: 1,1,1,1,1
↦ slimgb in ring (0),(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 (0),(b,c,d,a,@),(lp(3),dp(1),dp(1),C)
↦ [63:1]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 (0),(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)
↦ [1023:2]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);
```

See also: Section 5.1.127 [slimgb], page 218; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section 5.1.135 [stdhilb], page 225.

5.1.45 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 3.8.2 [Format of a library], page 54; Section 3.7.1 [Procedure definition], page 49; Section 3.1.3 [The online help system], page 15; Section 5.1.137 [system], page 227.

5.1.46 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 5.1.20 [dim], page 141; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.47 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 ideal, resp. 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 5.1.135 [stdhilb], page 225).

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 R=32003,(x,y,z),dp;
ideal i=x2,y2,z2;
ideal s=std(i);
hilb(s);
↦ //         1 t^0
↦ //        -3 t^2
↦ //         3 t^4
↦ //        -1 t^6
↦
↦ //         1 t^0
↦ //         3 t^1
↦ //         3 t^2
```

```
↦ //         1 t^3
↦ // 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
```

See Section C.2 [Hilbert function], page 507; Section 4.3 [ideal], page 72; Section 4.6 [intvec], page 85; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223; Section 5.1.135 [stdhilb], page 225.

5.1.48 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 ( 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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.49 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 = `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 5.1.3 [betti], page 130; Section 4.3 [ideal], page 72; Section 4.4 [int], page 77; Section 5.1.74 [lres], page 177; Section 5.1.82 [minres], page 184; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.118 [res], page 209; Section 5.1.131 [sres], page 221.

5.1.50 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 4.9 [map], page 98 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.

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 5.1.32 [fetch], page 148; Section 5.1.48 [homog], page 160; Section 4.9 [map], page 98; Section 4.16 [qring], page 117; Section 4.18 [ring], page 118.

5.1.51 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 5.1.117 [repart], page 208.

5.1.52 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 4.3 [ideal], page 72; Section 5.1.133 [std], page 223.

5.1.53 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 5.1.17 [delete], page 139; Section 4.8 [list], page 96.

5.1.54 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 5.1.19 [diff], page 140; Section 5.1.33 [fglm], page 149; Section 5.1.56 [intersect], page 165; Section 5.1.133 [std], page 223; Section 5.1.136 [subst], page 226.

5.1.55 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 or mixed 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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223.

5.1.56 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 `returnSB` is enabled then the result is a standard basis.

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));
  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
```

See Section 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.98 [option], page 192.

5.1.57 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+2x
   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 5.1.19 [diff], page 140; Section 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.96 [nvars], page 192.

5.1.58 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 5.1.44 [groebner], page 155; Section 4.3 [ideal], page 72; Section 5.1.133 [std], page 223.

5.1.59 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 )`
`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 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`.

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 5.1.15 [deg], page 138; Section 4.3 [ideal], page 72; Section 4.4 [int], page 77; Section 4.6 [intvec], page 85; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.60 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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.149 [vdim], page 234.

5.1.61 kernel

Syntax: `kernel` (ring_name, map_name)
`preimage` (ring_name, ideal_expression)

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 657; Section D.4.7.15 [hom_kernel], page 703; Section 4.3 [ideal], page 72; Section 4.9 [map], page 98; Section 5.1.83 [modulo], page 185; Section 5.1.104 [preimage], page 198; Section 4.18 [ring], page 118.

5.1.62 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 5.1.14 [defined], page 138; Section D.2.3 [general_lib], page 537.

5.1.63 killattrib

Syntax: `killattrib` (name, string_expression)

Type: none

Purpose: deletes 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
```

See Section 5.1.1 [attrib], page 127; Section 5.1.98 [option], page 192.

5.1.64 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 4.4 [int], page 77; Section 4.10 [matrix], page 101.

5.1.65 laguerre

Syntax: `laguerre` (poly_expression, int_expression, int_expression)

Type: list

Purpose: 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. The 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.

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,10,2);
↦ [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)
```

5.1.66 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 4.3 [ideal], page 72; Section 5.1.67 [leadcoef], page 172; Section 5.1.68 [leadexp], page 173; Section 5.1.69 [leadmonom], page 173; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.67 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 5.1.66 [lead], page 172; Section 5.1.68 [leadexp], page 173; Section 5.1.69 [leadmonom], page 173; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.68 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 4.6 [intvec], page 85; Section 5.1.66 [lead], page 172; Section 5.1.67 [leadcoef], page 172; Section 5.1.69 [leadmonom], page 173; Section 5.1.85 [monomial], page 186; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.69 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 4.6 [intvec], page 85; Section 5.1.66 [lead], page 172; Section 5.1.67 [leadcoef], page 172; Section 5.1.68 [leadexp], page 173; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.70 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 3.8.1 [Loading a library], page 53, 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 (12541,2010-02-09)
```

See Section 3.1.6 [Command line options], page 19; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 525; Section 5.2.11 [load], page 245; Section 4.13 [package], page 111; Section 4.15 [proc], page 116; Section D.1 [standard_lib], page 525; Section 4.19 [string], page 120; Section 5.1.137 [system], page 227.

5.1.71 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 )`

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.

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 5.1.21 [division], page 141; Section 4.3 [ideal], page 72; Section 4.11 [module], page 105.

5.1.72 liftstd

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 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 then `matrix(sm)=matrix(m)*T` and `sm=ideal(matrix(m)*T)`, resp. `sm=module(matrix(m)*T)`.
In an optional third argument the syzygy module will be returned.

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,7y6+xz, -7056y6-1008xz,
↦ 0,     -3x2-yz,3024x2-xy+1008yz-2z,
↦ 3x2+yz,0,      7y6+xz
```

See Section 4.3 [ideal], page 72; Section 4.10 [matrix], page 101; Section 5.1.98 [option], page 192; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223.

5.1.73 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 in the current namespace:

- `listvar()`: all currently visible names except procedures,
- `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.

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
↦ // 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
  LIB "poly.lib";
  listvar(Poly);
↦ // Poly                   [0]  package (S,poly.lib)
↦ // ::newtonDiag           [0]  proc from poly.lib
↦ // ::subrInterred         [0]  proc from poly.lib
↦ // ::id2mod               [0]  proc from poly.lib
↦ // ::mod2id               [0]  proc from poly.lib
↦ // ::denominator          [0]  proc from poly.lib
↦ // ::numerator            [0]  proc from poly.lib
↦ // ::content              [0]  proc from poly.lib
↦ // ::lcm                  [0]  proc from poly.lib
↦ // ::rad_con              [0]  proc from poly.lib
↦ // ::normalize            [0]  proc from poly.lib
↦ // ::mindeg1              [0]  proc from poly.lib
↦ // ::mindeg               [0]  proc from poly.lib
↦ // ::maxdeg1              [0]  proc from poly.lib
↦ // ::maxdeg               [0]  proc from poly.lib
↦ // ::maxcoef              [0]  proc from poly.lib
↦ // ::is_zero              [0]  proc from poly.lib
↦ // ::freerank             [0]  proc from poly.lib
↦ // ::kat_var              [0]  proc from poly.lib
↦ // ::katsura              [0]  proc from poly.lib
↦ // ::elemSymmId           [0]  proc from poly.lib
↦ // ::elemSymmPoly         [0]  proc from poly.lib
↦ // ::cyclic               [0]  proc from poly.lib
↦ // ::substitute           [0]  proc from poly.lib
↦ // ::hilbPoly             [0]  proc from poly.lib
↦ // ::bino                 [0]  proc from poly.lib (static)
```

See Section 3.5.3 [Names], page 42; Section 3.7.2 [Names in procedures], page 51; Section 5.1.14 [defined], page 138; Section 5.1.91 [names], page 189; Section 4.13 [package], page 111; Section 5.1.141 [type], page 231.

5.1.74 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

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} R \longrightarrow R/I \longrightarrow 0.$$

If the int_expression k is not zero then the computation stops after k steps and returns a list of modules M_i = `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 5.1.3 [betti], page 130; Section 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 4.4 [int], page 77; Section 5.1.82 [minres], page 184; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.118 [res], page 209; Section 5.1.131 [sres], page 221.

5.1.75 ludecomp

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 Section 5.1.76 [luinverse], page 179.

5.1.76 luinverse

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 Section 5.1.75 [ludecomp], page 178.

5.1.77 lusolve

Syntax: `lusolve (` matrix_expression`,` matrix_expression`,` matrix_expression`,` matrix_expression `)`

Type: matrix

Purpose: Computes a solution 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, i.e., a vector.

`list L=lusolve(P,L,U,b);` fills the list L with either one entry = 0 (signaling that A*x=b has no solution), or with the three entries 1, x, d, where x is any (n x 1) solution and d the dimension of the affine solution space.

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.

Example:

```
ring r=0,(x),dp;
matrix A[4][3]=1,1,3,2,1,4,3,0,3,4,0,4;
matrix b[4][1]=5,7,6,8;
list L=ludecomp(A);
list Q=lusolve(L[1],L[2],L[3],b);
if (Q[1] == 1)
{
  print(Q[2]);
  "----- next should be the zero vector:";
  print(A*Q[2]-b);
  "solution space has dimension", Q[3];
}
↦ 2,
↦ 3,
↦ 0
↦ ----- next should be the zero vector:
↦ 0,
↦ 0,
↦ 0,
↦ 0
↦ solution space has dimension 1
```

See Section 5.1.75 [ludecomp], page 178.

5.1.78 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 4.3 [ideal], page 72; Section 4.18 [ring], page 118.

5.1.79 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 5.1.98 [option], page 192).

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)/1023), "KByte)" +newline+
   "Memory currently allocated from system:",memory(1), "Byte (",
   int(memory(1)/1023), "KByte)";
   "Maximal memory allocated from system  :",memory(2), "Byte (",
   int(memory(2)/1023), "KByte)";
}
ReportMemoryUsage();
↦ Memory currently used by SINGULAR     : 224536 Byte ( 219 KByte)
↦ Memory currently allocated from system: 670284 Byte ( 655 KByte)
↦ Maximal memory allocated from system  : 702092 Byte ( 686 KByte)
kill p;
ReportMemoryUsage(); // less memory used: p killed
```

```
↦ Memory currently used by SINGULAR     : 171296 Byte ( 167 KByte)
↦ Memory currently allocated from system: 670284 Byte ( 655 KByte)
↦ Maximal memory allocated from system  : 702092 Byte ( 686 KByte)
  kill r;
  ReportMemoryUsage(); // even less memory: r killed
↦ Memory currently used by SINGULAR     : 160648 Byte ( 157 KByte)
↦ Memory currently allocated from system: 666272 Byte ( 651 KByte)
↦ Maximal memory allocated from system  : 702092 Byte ( 686 KByte)
```

See Section 5.1.98 [option], page 192; Section 5.1.137 [system], page 227.

5.1.80 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.

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 5.1.88 [mstd], page 187.

5.1.81 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.

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 5.1.18 [det], page 140.

5.1.82 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
↦ resolution not minimized yet
↦
  list(rs);
↦ [1]:
↦    _[1]=x3+x2y+xy2+y3
↦    _[2]=xy4
↦    _[3]=y7
↦ [2]:
↦    _[1]=-y4*gen(1)+x2*gen(2)+xy*gen(2)+y2*gen(2)+gen(3)
↦    _[2]=-y3*gen(2)+x*gen(3)
  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 5.1.87 [mres], page 186; Section 5.1.118 [res], page 209; Section 5.1.131 [sres], page 221.

5.1.83 modulo

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 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.

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.4.7.15 [hom_kernel], page 703; Section 5.1.138 [syz], page 229.

5.1.84 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 4.7.2 [link expressions], page 88.

5.1.85 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 4.6 [intvec], page 85; Section 5.1.68 [leadexp], page 173.

5.1.86 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 5.1.144 [uressolve], page 232.

5.1.87 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 = `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 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 5.1.74 [lres], page 177; Section 4.11 [module], page 105; Section 5.1.118 [res], page 209; Section 5.1.131 [sres], page 221.

5.1.88 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 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 4.3 [ideal], page 72; Section 5.1.80 [minbase], page 182; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223.

5.1.89 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 5.1.16 [degree], page 139; Section 5.1.20 [dim], page 141; Section 4.3 [ideal], page 72; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

5.1.90 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 5.1.91 [names], page 189; Section 5.1.119 [reservedName], page 210; Section 5.1.142 [typeof], page 231.

5.1.91 names

Syntax: `names ( )`
`names (` ring_name `)`
`names (` package_name `)`

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.
package_name can be `Current`, `Top` or any other identifier of type package.

Example:

```
  int i=9;
  ring r;
  poly f;
  package p;
  int 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
```

See Section 5.1.90 [nameof], page 188; Section 5.1.119 [reservedName], page 210.

5.1.92 ncols

Syntax: `ncols ( matrix_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 4.10 [matrix], page 101; Section 5.1.95 [nrows], page 191; Section 5.1.126 [size], page 217.

5.1.93 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 5.1.101 [par], page 197; Section 5.1.103 [parstr], page 198; Section 4.18 [ring], page 118.

5.1.94 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 $coker(A_1) = F_0/M$

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0,$$

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 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 5.1.74 [lres], page 177; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.118 [res], page 209; Section 4.17 [resolution], page 117; Section 5.1.131 [sres], page 221.

5.1.95 nrows

Syntax:
```
nrows ( matrix_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 5.1.42 [gen], page 154; Section 4.10 [matrix], page 101; Section 4.11 [module], page 105; Section 5.1.92 [ncols], page 190; Section 4.20 [vector], page 124.

5.1.96 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 5.1.93 [npars], page 190; Section 4.18 [ring], page 118; Section 5.1.146 [var], page 233; Section 5.1.148 [varstr], page 234.

5.1.97 open

Syntax: `open ( link_expression )`

Type: none

Purpose: opens a link.

Example:

```
link l="MPtcp:launch";
open(l);  // start SINGULAR "server" on localhost in batchmode
close(l); // shut down SINGULAR server
```

See Section 5.1.9 [close], page 133; Section 4.7 [link], page 88.

5.1.98 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).

`returnSB` the functions `syz`, `intersect`, `quotient`, `modulo` 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 reduceers in std computations,

`notRegularity`
disables the regularity bound for `res` and `mres` (see Section 5.1.116 [regularity], page 208).

`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`, `lres`, `mres`, `minres`, `mstd`, `res`, `slimgb`, `sres`, `std`, `stdfglm`, `stdhilb`, `syz`. See below for more details.

`qringNF` simplifies always modulo the current `qring`.

`redSB` computes a reduced standard basis in any standard basis computation.

`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.

`redThrough`
for inhomogenous 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.

`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.

`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.

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 249).

`degBound` a degree bound is set (see Section 5.3.1 [degBound], page 247).

The last set of options controls the output of SINGULAR:

`Imap` shows the mapping of variables with the fetch and imap 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 5.1.79 [memory], page 181).

`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
	`.`	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
	`(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 247; Section 5.3.4 [multBound], page 249; Section 5.1.133 [std], page 223.

5.1.99 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 5.1.15 [deg], page 138; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.100 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 5.1.6 [charstr], page 132; Section 5.1.103 [parstr], page 198; Section 4.18 [ring], page 118; Section 5.1.148 [varstr], page 234.

5.1.101 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 5.1.4 [char], page 131; Section 5.1.93 [npars], page 190; Section 5.1.103 [parstr], page 198; Section 4.18 [ring], page 118; Section 5.1.146 [var], page 233.

5.1.102 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 5.1.15 [deg], page 138; Section 4.12 [number], page 108; Section 4.18 [ring], page 118; Section 5.1.146 [var], page 233.

5.1.103 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 5.1.6 [charstr], page 132; Section 5.1.93 [npars], page 190; Section 5.1.100 [ordstr], page 197; Section 4.18 [ring], page 118; Section 5.1.148 [varstr], page 234.

5.1.104 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, whose generator is not in the image of the map, is the zero ideal.

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 4.3 [ideal], page 72; Section 5.1.61 [kernel], page 169; Section 4.9 [map], page 98; Section 4.18 [ring], page 118.

5.1.105 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 537; Section 4.4 [int], page 77.

5.1.106 primefactors

Syntax: `primefactors ( bigint_expression , [ int_expression ] )`

Syntax: `primefactors ( 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 a bigint, no ring needs to be active.
The method is capable of finding all prime factors <= 2^31 - 1, i.e., it finds all prime factors <= min(2^31-1, b). The quotient, q, arising by dividing out all found prime factors will also be returned. More concretely, a list L with three entries is returned:
L[1] = q,
L[2][i] = i-th prime factor (in ascending order),
L[3][i] = multiplicity of L[2][i].

Example:

```
  bigint n = bigint(7)^12 * bigint(37)^6 * 121;
  primefactors(n);
↦ [1]:
↦    429706768884528633628 9
↦ [2]:
↦    [1]:
↦       7
↦    [2]:
↦       11
↦    [3]:
↦       37
↦ [3]:
↦    [1]:
↦       12
↦    [2]:
↦       2
↦    [3]:
↦       6
  primefactors(n, 15); /* only prime factors <= 15 */
↦ [1]:
↦    429706768884528633628 9
↦ [2]:
↦    [1]:
↦       7
↦    [2]:
↦       11
↦    [3]:
↦       37
↦ [3]:
↦    [1]:
↦       12
↦    [2]:
↦       2
↦    [3]:
↦       6
```

See Section 5.1.105 [prime], page 199.

5.1.107 print

Syntax: `print` (expression)
`print` (expression, `"betti"`)
`print` (expression, format_string)

Type: none (for the first two calling sequences), resp.
string (for the last calling sequence)

Purpose: The first form prints the expression to the terminal and has no return value. Use the format string `%p` to print into a string (see below).
The second form prints the graded Betti numbers from a matrix and has no return value. 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. The format string determines which 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                    [0]  module, rk 2, 2 generator(s)
↦ 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;
↦ (0),(x,y,z),(dp(3),C),1,1
  s=print(l,"%2s"); s;
↦ (0),(x,y,z),(dp(3),C),
↦ 1,1
↦
  s=print(l,"%l"); s;
↦ list("(0),(x,y,z),(dp(3),C)",intmat(intvec(1,1 ),1,2))
```

See Section 3.5.5 [Type conversion and casting], page 44; Section 5.1.3 [betti], page 130; Section 5.1.13 [dbprint], page 137; Section 5.1.38 [fprintf], page 152; Section 5.1.108 [printf], page 202; Section 5.3.7 [short], page 250; Section 5.1.130 [sprintf], page 220; Section 4.19.3 [string type cast], page 122; Section 5.1.141 [type], page 231.

5.1.108 printf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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:(0),(x,y,z),(dp(3),C)
printf("s:%s",list(l));
↦ s:(0),(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("(0),(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 5.1.38 [fprintf], page 152; Section 5.1.107 [print], page 200; Section 5.1.130 [sprintf], page 220; Section 4.19 [string], page 120.

5.1.109 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"`.

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 4.11 [module], page 105.

5.1.110 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 4.3 [ideal], page 72; Section 4.6 [intvec], page 85; Section 5.1.151 [weight], page 235.

5.1.111 quote

Syntax: `quote ( expression )`

Type: none

Purpose: prevents expressions from evaluation. Used only in connections with write to MPfile 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="MPfile:w example.mp";
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);
↦ [1023:1]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 4.7.5.1 [MPfile links], page 91; Section 5.1.24 [eval], page 144; Section 5.1.153 [write], page 236.

5.1.112 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 $\mathtt{R}^n$. 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 5.1.34 [fglmquot], page 150; Section 4.3 [ideal], page 72; Section 4.11 [module], page 105.

5.1.113 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.

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 4.4 [int], page 77; Section 4.5 [intmat], page 82; Section 5.1.137 [system], page 227.

5.1.114 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 MP links, one expression is read from the link and returned after evaluation. See Section 4.7.5 [MP links], page 90.
For MPtcp 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 4.7.6 [DBM links], page 94.

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 5.1.27 [execute], page 145; Section 5.1.43 [getdump], page 155; Section 4.7 [link], page 88; Section 5.1.132 [status], page 222; Section 5.1.153 [write], page 236.

5.1.115 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 reduce also with bad ecart (in the local case)

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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223; Section 4.20 [vector], page 124.

5.1.116 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 $0 \to \bigoplus_a K[x]e_{a,n} \to \ldots \to \bigoplus_a K[x]e_{a,0} \to I \to 0$ 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 4.8 [list], page 96; Section 5.1.82 [minres], page 184; Section 5.1.87 [mres], page 186; Section 5.1.98 [option], page 192; Section 5.1.118 [res], page 209; Section 4.17 [resolution], page 117; Section 5.1.131 [sres], page 221.

5.1.117 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 5.1.51 [impart], page 162.

5.1.118 res

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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 5.1.94 [nres], page 190.

homogeneous ideals and k=0:
`lres` (La'Scala's method), see Section 5.1.74 [lres], page 177.

not minimized resolution and (homogeneous input with k not 0, or local rings):
`sres` (Schreyer's method), see Section 5.1.131 [sres], page 221.

all other inputs:
`mres` (classical method), see Section 5.1.87 [mres], page 186.

Note: Accessing single elements of a resolution may require some partial computations to be finished and may therefore take some time.

See also Section 5.1.3 [betti], page 130; Section 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 5.1.74 [lres], page 177; Section 5.1.82 [minres], page 184; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.94 [nres], page 190; Section 4.17 [resolution], page 117; Section 5.1.131 [sres], page 221.

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
↦ resolution not minimized yet
↦
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
↦
```

5.1.119 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).

Example:

```
reservedName();
↦ ... // output skipped
  reservedName("ring");
↦ 1
  reservedName("xyz");
↦ 0
```

See Section 5.1.91 [names], page 189; Section 4.19 [string], page 120.

5.1.120 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 4.14 [poly], page 112; Section 4.18 [ring], page 118.

5.1.121 ringlist

Syntax: `ringlist ( ring_expression )`
`ringlist ( qring_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 302) components:

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).

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));
def ra = ring(l);     //creates the newring
ra; setring ra;
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   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
list lra = ringlist(ra);
lra[1][4]=ideal(a2+1);
def Ra = ring(lra);
setring Ra; Ra;
↦ //   characteristic : 0
↦ //   1 parameter    : a
```

```
↦ //   minpoly        : (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 4.16 [qring], page 117; Section 4.18 [ring], page 118.

5.1.122 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 5.1.14 [defined], page 138; Section 4.18 [ring], page 118; Section 5.1.146 [var], page 233; Section 5.1.148 [varstr], page 234.

5.1.123 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 5.2.10 [keepring], page 244; Section 4.16 [qring], page 117; Section 4.18 [ring], page 118.

5.1.124 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 occured;
* 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.7.2.8 [simplexOut], page 1041 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.7.2.8 [simplexOut], page 1041.

5.1.125 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 rule given as the second argument. The simplification rules are the sum of the following functions:

1	normalize (make leading coefficients 1).
2	erase zero generators/columns.
4	keep only the first one of identical generators/columns.
8	keep only the first one of generators/columns which differ only by a factor in the ground field.
16	keep only those generators/columns whose leading monomials differ.
32	keep only those generators/columns whose leading monomials are not divisible by other ones.

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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

5.1.126 size

Syntax: `size` (string_expression)
`size` (intvec_expression)
`size` (intmat_expression)
`size` (poly_expression)
`size` (vector_expression)
`size` (ideal_expression)
`size` (module_expression)
`size` (matrix_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, list or resolution
: returns the length, i.e., the number of characters, entries or elements.

poly or vector
: returns the number of monomials.

matrix 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

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 4.3 [ideal], page 72; Section 4.5 [intmat], page 82; Section 4.6 [intvec], page 85; Section 4.11 [module], page 105; Section 5.1.92 [ncols], page 190; Section 5.1.95 [nrows], page 191; Section 4.14 [poly], page 112; Section 4.19 [string], page 120; Section 4.20 [vector], page 124.

5.1.127 slimgb

Syntax:

`slimgb` (ideal_expression)
`slimgb` (module_expression)

Type: ideal or module

Purpose: Section A.2.3 [slim Groebner bases], page 449

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 `http://www.mathematik.uni-kl.de/~zca/Reports_on_ca/35/paper_35_full.ps.gz`. A reduced Groebner basis is returned if option(redSB) is set (see [option(redSB)], page 193). To view the progress of long running computations, use option(prot) (see [option(prot)], page 193).

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 5.1.44 [groebner], page 155; Section 4.3 [ideal], page 72; Section 5.1.98 [option], page 192; Section 4.18 [ring], page 118; Section 5.1.133 [std], page 223.

5.1.128 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 537.

5.1.129 sqrfree

Syntax: `sqrfree ( poly_expression )`

Type: ideal

Purpose: computes a squarefree decomposition of the given polynomial.

Example:

```
ring r=3,(x,y,z),dp;
poly f=(x-y)^3*(x+z)*(y-z);
sqrfree(f);
↦ _[1]=-xy+xz-yz+z2
```

See Section 5.1.30 [factorize], page 147.

5.1.130 sprintf

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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:
↦ (0),(x,y,z),(dp(3),C)
s = sprintf("s:%2%s", list(l)); s;
↦ s:
↦ (0),(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("(0),(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]:
↦    //   characteristic : 0
↦ //   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]:
↦    //   characteristic : 0
↦ //   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 5.1.38 [fprintf], page 152; Section 5.1.107 [print], page 200; Section 5.1.108 [printf], page 202; Section 4.19 [string], page 120.

5.1.131 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 $coker(A_1) = F_0/M$

$$... \longrightarrow F_2 \xrightarrow{A_2} F_1 \xrightarrow{A_1} F_0 \longrightarrow F_0/M \longrightarrow 0.$$

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 5.1.3 [betti], page 130; Section 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 4.4 [int], page 77; Section 5.1.74 [lres], page 177; Section 5.1.82 [minres], page 184; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.118 [res], page 209; Section 5.1.138 [syz], page 229.

5.1.132 status

Syntax: `status ( link_expression, string_expression )`

Type: string

Syntax: `status ( link_expression, string_expression, string_expression )`
`status ( link_expression, string_expression, string_expression, int_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)`. If a fourth integer argument (say `i`) is given and if `status(l,s1,s2)` yields `0`, then the execution of the current process is suspended (the process is put to "sleep") for approximately `i` microseconds, and afterwards the result of another call to `status(l,s1,s2)` is returned. The latter is useful for "polling" the `read` status of MPtcp links such that busy loops are avoided (see Section A.1.8 [Parallelization with MPtcp links], page 443 for an example). Note that on some systems, the minimum time for a process to be put to sleep is one second. The following string expressions are allowed:

`"name"` the name string given by the definition of the link (usually the filename)

`"type"` returns `"ASCII"`, `"MPfile"`, `"MPtcp"` or `"DBM"`

`"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/MPfile links

Example:

```
link l=":w example.txt";
status(l,"write");
↦ not ready
open(l);
status(l,"write","ready");
↦ 1
close(l);
```

See Section 4.7 [link], page 88; Section 5.1.97 [open], page 192; Section 5.1.114 [read], page 206; Section 5.1.153 [write], page 236.

5.1.133 std

Syntax:

```
std ( ideal_expression)
std ( module_expression)
std ( ideal_expression, intvec_expression )
std ( module_expression, intvec_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 or module

Purpose: returns a standard basis of an ideal or module 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 an optional second argument of type intvec as Hilbert series (result of `hilb(i,1)`, see Section 5.1.47 [hilb], page 159), if the ideal, resp. module, is homogeneous (Hilbert driven standard basis computation, Section 5.1.135 [stdhilb], page 225). 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`.

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 5.1.44 [groebner], page 155), 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 193).

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 5.1.29 [facstd], page 146; Section 5.1.33 [fglm], page 149; Section 5.1.44 [groebner], page 155; Section 4.3 [ideal], page 72; Section 5.1.88 [mstd], page 187; Section 5.1.98 [option], page 192; Section 4.18 [ring], page 118; Section 5.1.134 [stdfglm], page 224; Section 5.1.135 [stdhilb], page 225.

5.1.134 stdfglm

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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 5.1.33 [fglm], page 149; Section 5.1.44 [groebner], page 155; Section 5.1.127 [slimgb], page 218; Section 5.1.133 [std], page 223; Section 5.1.135 [stdhilb], page 225.

5.1.135 stdhilb

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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 5.1.44 [groebner], page 155; Section 5.1.127 [slimgb], page 218; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224.

5.1.136 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.

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 4.3 [ideal], page 72; Section 4.9 [map], page 98; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section D.2.5.19 [substitute], page 559; Section 4.20 [vector], page 124.

5.1.137 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("sh"`, string_expression)
: shell escape, returns the return code of the shell as int. The string is sent literally to the shell.

`system("pid")`
: returns the process number as int (for creating unique names).

`system("cpu")`
: returns the number of cpu cores as int (for using multiple cores).

`system("uname")`
: returns a string identifying the architecture for which SINGULAR was compiled.

`system("getenv",` string_expression)
: returns the value of the shell environment variable given as the second argument. The return type is string.

`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("tty")`
: resets the terminal.

`system("version")`
: returns the version number of SINGULAR as int.

`system("contributors")`
: returns names of people who contributed to the SINGULAR kernel as string.

`system("gen")`
: returns the generating element of the multiplicative group of (Z/p)\{0} (as int) where p is the characteristic of the basering.

`system("nblocks")`

`system("nblocks", ring_name )`
: returns the number of blocks of the given ring, or the number of parameters of the current basering, if no second argument is given. The return type is int.

`system("Singular")`
: returns the absolute (path) name of the running SINGULAR as string.

`system("SingularLib")`
: returns the colon seperated library search path name as string.

`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.

`system("browsers");`
: returns a string about available help browsers. See Section 3.1.3 [The online help system], page 15. returns the number of cpus as int (for creating multiple threads/processes).

`system("pid")`

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
↦ // --quiet           1
↦ // --sort            0
↦ // --random          12345678
↦ // --no-tty          1
↦ // --user-option
↦ // --allow-net       0
↦ // --browser
↦ // --emacs           0
↦ // --no-stdlib       0
↦ // --no-rc           1
↦ // --no-warn         0
↦ // --no-out          0
↦ // --min-time        "0.5"
↦ // --MPport
↦ // --MPhost
↦ // --MPrsh
↦ // --ticks-per-sec   1
↦ // --MPtransp
↦ // --MPmode
// 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 netscape to access HTML pages from the net
system("--allow-net", 1);
// and set help browser to netscape
system("--browser", "netscape");
↦ // ** Could not get IdxFile.
↦ // ** Either set environment variable SINGULAR_IDX_FILE to IdxFile,
↦ // ** or make sure that IdxFile is at /home/hannes/singular/doc/singular
   idx
↦ // ** ressource `x` not found
↦ // ** Setting help browser to 'builtin'.
```

5.1.138 syz

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.

Example:

```
ring R=0,(x,y),(c,dp);
ideal i=x,y;
syz(i);
↦ _[1]=[y,-x]
```

See Section 5.1.49 [hres], page 161; Section 4.3 [ideal], page 72; Section 5.1.74 [lres], page 177; Section 4.11 [module], page 105; Section 5.1.87 [mres], page 186; Section 5.1.94 [nres], page 190; Section 5.1.118 [res], page 209; Section 5.1.131 [sres], page 221.

5.1.139 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 4.5 [intmat], page 82; Section 4.10 [matrix], page 101.

5.1.140 transpose

Syntax: `transpose ( intmat_expression )`
`transpose ( matrix_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 4.5 [intmat], page 82; Section 4.10 [matrix], page 101; Section 4.11 [module], page 105.

5.1.141 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                        [0]  int 3
```

See Chapter 4 [Data types], page 70; Section 5.1.73 [listvar], page 176; Section 5.1.107 [print], page 200.

5.1.142 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(_);
↦ ?unknown type?
  type i;
↦ // i                        [0]  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 70; Section 5.1.141 [type], page 231.

5.1.143 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 5.1.68 [leadexp], page 173; Section 5.1.146 [var], page 233.

5.1.144 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 Minkowsky 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 5.1.65 [laguerre], page 171; Section 5.1.86 [mpresmat], page 186.

5.1.145 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
```

5.1.146 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 4.4 [int], page 77; Section 5.1.96 [nvars], page 192; Section 4.18 [ring], page 118; Section 5.1.143 [univariate], page 232; Section 5.1.148 [varstr], page 234.

5.1.147 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 5.1.68 [leadexp], page 173; Section 5.1.96 [nvars], page 192; Section 5.1.143 [univariate], page 232; Section 5.1.146 [var], page 233; Section 5.1.148 [varstr], page 234.

5.1.148 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 5.1.6 [charstr], page 132; Section 4.4 [int], page 77; Section 5.1.96 [nvars], page 192; Section 5.1.100 [ordstr], page 197; Section 5.1.103 [parstr], page 198; Section 4.18 [ring], page 118; Section 5.1.146 [var], page 233.

5.1.149 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.5.13.1 [codim], page 962; Section 5.1.16 [degree], page 139; Section 5.1.20 [dim], page 141; Section 4.3 [ideal], page 72; Section 5.1.60 [kbase], page 168; Section 5.1.89 [mult], page 188; Section 5.1.133 [std], page 223.

5.1.150 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 4.4 [int], page 77; Section 4.10 [matrix], page 101; Section 5.1.81 [minor], page 182.

5.1.151 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 4.3 [ideal], page 72; Section 4.6 [intvec], page 85; Section 5.1.110 [qhweight], page 203.

5.1.152 weightKB

Procedure from library `standard.lib` (see Section D.1 [standard_lib], page 525).

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 5.1.60 [kbase], page 168.

5.1.153 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 MP 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 MPtcp 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 MP format and MPfile links are better:
write("MPfile:w save_i.mp",i);
def j=read("MPfile:r save_i.mp");
```

See Chapter 4 [Data types], page 70; Section 5.1.22 [dump], page 142; Section 5.1.24 [eval], page 144; Section 4.7 [link], page 88; Section 5.1.107 [print], page 200; Section 5.1.108 [printf], page 202; Section 5.1.111 [quote], page 204; Section 5.1.114 [read], page 206; Section 5.3.7 [short], page 250.

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 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 237; Section 5.2.7 [for], page 242; Section 5.2.14 [while], page 246.

5.2.2 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 850 in groebner
breakpoint(groebner, 176);
↦ breakpoint 2, at line 176 in groebner
breakpoint(groebner, -1);
↦ breakpoints in groebner deleted(0x6)
```

See Section 3.10.2 [Source code debugger], page 66; Section 5.2.15 [~], page 247.

5.2.3 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 237; Section 5.2.7 [for], page 242; Section 5.2.14 [while], page 246.

5.2.4 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 237; Section 4.4.5 [boolean expressions], page 80; Section 5.2.8 [if], page 242.

5.2.5 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 5.2.6 [exportto], page 240; Section 5.2.9 [importfrom], page 242; Section 5.2.10 [keepring], page 244.

5.2.6 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 transfered to the other package. It does no longer exist in the current package. If the identifier should only be copied, Section 5.2.9 [importfrom], page 242 should be used instead.

Example:

```
proc p1
{
  int i,j;
  exportto(Current,i);
  intmat m;
  listvar(Current);
  exportto(Top,m);
}
p1();
↦ // Top                    [0]  package (N)
↦ // ::m                    [1]  intmat 1 x 1
↦ // ::i                    [0]  int 0
↦ // ::j                    [1]  int 0
↦ // ::#                    [1]  list, size: 0
↦ // ::p1                   [0]  proc
↦ // ::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
package Test1;
exportto(Test1,p1);
listvar(Top);
↦ // Top                    [0]  package (N)
↦ // ::m                    [0]  intmat 1 x 1
↦ // ::i                    [0]  int 0
↦ // ::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
listvar(Test1);
↦ // Test1                [0]  package (N)
↦ // ::p1                 [0]  proc
Test1::p1();
↦ // Test1                [0]  package (N)
↦ // ::m                  [1]  intmat 1 x 1
↦ // ::i                  [0]  int 0
↦ // ::j                  [1]  int 0
↦ // ::#                  [1]  list, size: 0
↦ // ::p1                 [0]  proc
↦ // ** redefining m
listvar(Top);
↦ // Top                  [0]  package (N)
↦ // ::m                  [0]  intmat 1 x 1
↦ // ::i                  [0]  int 0
↦ // ::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
listvar(Test1);
↦ // Test1                [0]  package (N)
↦ // ::i                  [0]  int 0
```

```
↦ // ::p1                    [0]  proc
```

See Section 5.2.5 [export], page 239; Section 5.2.9 [importfrom], page 242; Section 5.2.10 [keepring], page 244.

5.2.7 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 237; Section 4.4.5 [boolean expressions], page 80; Section 5.2.1 [break], page 237; Section 5.2.3 [continue], page 238; Section 5.2.8 [if], page 242; Section 5.2.14 [while], page 246.

5.2.8 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 237; Section 4.4.5 [boolean expressions], page 80; Section 5.2.1 [break], page 237; Section 5.2.4 [else], page 239.

5.2.9 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 5.2.6 [exportto], page 240 should be used instead.

Example:

```
listvar(Top);
↦ // Top                    [0]  package (N)
↦ // ::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
load("inout.lib");
listvar(Top);
↦ // Top                    [0]  package (N)
↦ // ::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
importfrom(Inout,pause);
listvar(Top);
```

```
↦ // Top                    [0]  package (N)
↦ // ::pause                [0]  proc from inout.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
```

See Section 5.2.5 [export], page 239; Section 5.2.6 [exportto], page 240; Section 5.2.10 [keepring], page 244.

5.2.10 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 4.18 [ring], page 118.

5.2.11 load

Syntax: `load(` string_expression `);`

Type: none

Purpose: reads a library of procedures from a file. In contrast to the command `LIB`, 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");`.

All loaded libraries are displayed by the `listvar(package);` command:

```
option(loadLib);   // show loading of libraries;
                   // standard.lib is loaded
listvar(package);
↦ // Standard             [0]  package (S,standard.lib)
↦ // Top                  [0]  package (N)
                   // the names of the procedures of inout.lib
load("inout.lib"); // are now known to Singular
↦ // ** loaded inout.lib (12541,2010-02-09)
listvar(package);
↦ // Inout                [0]  package (S,Inout)
↦ // Standard             [0]  package (S,standard.lib)
↦ // Top                  [0]  package (N)
```

Each time a library (Section 3.8 [Libraries], page 53) / dynamic module (Section 3.11 [Dynamic loading], page 68) 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 (12541,2010-02-09)
listvar();
```

See Section 3.1.6 [Command line options], page 19; Section A.1.9 [Dynamic modules], page 444; Section 5.1.70 [LIB], page 174; Section 2.3.3 [Procedures and libraries], page 10; Appendix D [SINGULAR libraries], page 525; Section 5.2.6 [exportto], page 240; Section 5.2.9 [importfrom], page 242; Section 4.13 [package], page 111; Section 4.15 [proc], page 116; Section D.1 [standard_lib], page 525; Section 4.19 [string], page 120; Section 5.1.137 [system], page 227.

5.2.12 quit

Syntax: `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;
```

See [exit], page 246.

5.2.13 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 70; Section 4.15 [proc], page 116.

5.2.14 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 237; Section 4.4.5 [boolean expressions], page 80; Section 5.2.1 [break], page 237.

5.2.15 ~ (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.

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 3.10.3 [Break points], page 67.

5.3 System variables

5.3.1 degBound

Type: int

Purpose: The standard basis computation is stopped if the total (weighted) degree exceeds `degBound`.
`degBound` should not be used for a global ordering with inhomogeneous input.
Reset this bound by setting `degBound` to 0.
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 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 5.1.15 [deg], page 138; Section 4.4 [int], page 77; Section 5.1.98 [option], page 192; Section 5.1.133 [std], page 223.

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 4.4 [int], page 77; Section 5.3.11 [voice], page 253.

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 5.1.30 [factorize], page 147).

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 5.1.30 [factorize], page 147; Section 4.18 [ring], page 118.

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);
multBound=0;  //disables multBound
j=std(i);
degree(j);
```

See Section 4.4 [int], page 77; Section 5.1.89 [mult], page 188; Section 5.1.98 [option], page 192; Section 5.1.133 [std], page 223.

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 4.14 [poly], page 112; Section 5.1.133 [std], page 223.

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 3.9.1 [Procedures in a library], page 55, 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 5.1.13 [dbprint], page 137; Section 4.4 [int], page 77; Section 5.3.11 [voice], page 253.

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".

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 4.4 [int], page 77.

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 253, 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/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
↦ 110
```

See Section 3.1.6 [Command line options], page 19; Section 5.3.6 [printlevel], page 249; Section 5.3.10 [rtimer], page 253; Section 5.1.137 [system], page 227; Section 5.3.11 [voice], page 253.

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` 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 4.4 [int], page 77.

5.3.10 rtimer

Type: int

Purpose: identical to `timer` (see Section 5.3.8 [timer], page 250), 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 3.9 [Guidelines for writing a library], 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 5.1.13 [dbprint], page 137; Section 5.1.73 [listvar], page 176; Section 5.3.6 [printlevel], page 249.

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 2147483629 (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.)
- 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.
- 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.

6.2 System dependent limitations

Ports of SINGULAR to different systems do not always implement all possible parts of SINGULAR:

- MP links are available only for 32bit unix systems.
- dynamic modules are implemented for
  - unix systems with ELF format for executables (Linux, Solaris)
  - HPUX

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 5.1.62 [kill], page 170).
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 5.2.10 [keepring], page 244) 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 apllies for large numerical constants which are of type `bigint`.

   ```
     ring r=32002,x,dp;
     3/2;    // ring division
   ↦ -15994
     3 / 2;  // integer division
   ↦ 1
     3 div 2;
   ↦ 1
     number(3) / number(2);
   ↦ -15994
     number a=3;
     number b=2;
     a/b;
   ↦ -15994
     int c=3;
     int d=2;
     c / d;
   ↦ 1
   ```

2. monomials and precedence

   The formation of a monomial has precedence over all operators:

   ```
     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 5.1.115 [reduce], page 206).

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 5.1.152 [weightKB], page 236)

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 K::varok;
string K::donotwork(1);
int K::i(1..3);
// Toplevel does not contain i(1..3)
listvar();
↦ // i(3)                 [0]  int 0
↦ // i(2)                 [0]  int 0
↦ // i(1)                 [0]  int 0
↦ // donotwork(1)         [0]  string
// i(1..3) are stored in Package 'K'
listvar(K);
↦ // K                    [0]  package (N)
↦ // ::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 variable,
5. a global ring variable (w.r.t. the current basering)
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

7.1 PLURAL

What is and what does PLURAL?

PLURAL is a kernel extension of SINGULAR, providing many algorithms for computations within certain non-commutative algebras (see Section 7.4 [Mathematical background (plural)], page 310 for detailed information on algebras and algorithms).

It uses the same data structures, 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.

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 310 for step-by-step building instructions and also Section 7.7 [Non-commutative libraries], page 320 for procedures for setting many important algebras easily).

Functionalities of PLURAL (enlisted in Section 7.3 [Functions (plural)], page 279) are accessible as soon as the basering becomes non-commutative (see Section 7.3.16 [nc_algebra], page 292).

One can perform various computations with polynomials and ideals in A and with vectors and submodules of a free module A^n.

One can work also within factor algebras of G -algebras (see Section 7.2.5 [qring (plural)], page 274 type) by two-sided ideals (see Section 7.3.29 [twostd], page 309).

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 due to Section 7.6 [LETTERPLACE], page 318).

In PLURAL one can only work with G -algebras and with their factor-algebras by two-sided ideals (GR -algebras).

PLURAL requires a monomial ordering but it does not work generally with local and mixed orderings. Right now, one can use only global orderings in PLURAL (see Section B.2.2 [General definitions for orderings], page 502), save for SCA, where we provide the possibility of computations in a tensor product of a non-commutative algebra (with a global ordering) with a commutative algebra (with any ordering). In the future, this will be enhanced for other algebras, as well.

PLURAL does not handle non-commutative parameters.

Defining parameters, one **cannot** impose non-commutative relations on them. Moreover, it is impossible to introduce parameters which do not commute with variables.

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 273.
Note, that PLURAL output consists only of monomials, hence the signs `*` are omitted.

`ideal` **(plural)**

Unless stated otherwise, an `ideal` as understood by PLURAL, is a list of generators of a **left** ideal. For more information see Section 7.2.1 [ideal (plural)], page 263.
For a **two-sided ideal** `T`, use command Section 7.3.29 [twostd], page 309 in order to compute the two-sided Groebner basis of `T`.

`module` **(plural)**

Unless stated otherwise, a `module` as understood by PLURAL, is **either** a fininitely 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 269 for details).

`qring` **(plural)**

In PLURAL it is only possible to build factor-algebras modulo **two-sided** ideals (see Section 7.2.5 [qring (plural)], page 274).

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 39.

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], page 309.

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 5.1.125 [simplify], page 216.

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 281)

`eliminate` elimination of variables (see Section 7.3.5 [eliminate (plural)], page 283)

`intersect` ideal intersection (see Section 7.3.9 [intersect (plural)], page 286)

`kbase` vector space basis of basering modulo the leading ideal (see Section 7.3.10 [kbase (plural)], page 287)

`lead` leading terms of a set of generators (see Section 5.1.66 [lead], page 172)

`lift` lift-matrix (see Section 7.3.11 [lift (plural)], page 288)

`liftstd` left Groebner basis and transformation matrix computation (see Section 7.3.12 [liftstd (plural)], page 289)

`maxideal` generators of a power of the maximal ideal at 0 (see Section 5.1.78 [maxideal], page 181)

`modulo` represents $(h1+h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 7.3.14 [modulo (plural)], page 291)

`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 291)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`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 295)

`oppose` creates an opposite ideal of a given ideal from the given ring into a basering (see Section 7.3.19 [oppose], page 296)

`preimage` preimage under a ring map (see Section 7.3.21 [preimage (plural)], page 299)

`quotient` ideal quotient (see Section 7.3.22 [quotient (plural)], page 299)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 301)

`simplify` simplify a set of polynomials (see Section 5.1.125 [simplify], page 216)

`size` number of non-zero generators (see Section 5.1.126 [size], page 217)

`slimgb` left Groebner basis computation with slim technique (see Section 7.3.25 [slimgb (plural)], page 304)

`std` left Groebner basis computation (see Section 7.3.26 [std (plural)], page 306)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 308)

`syz` computation of the first syzygy module (see Section 7.3.28 [syz (plural)], page 308)

`twostd` two-sided Groebner basis computation (see Section 7.3.29 [twostd], page 309)

`vdim` vector space dimension of basering modulo the leading ideal (see Section 7.3.30 [vdim (plural)], page 310)

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 260, Section 3.7.2 [Names in procedures], page 51.

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 267.

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 a map and not a morphism
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 algebra 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 285; Section 7.2.1.2 [ideal expressions (plural)], page 264; Section 7.3.8 [imap (plural)], page 285; Section 7.2.2 [map (plural)], page 267; Section 7.2.7 [ring (plural)], page 277.

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 285)

`imap (plural)`
a convenient map procedure for inclusions and projections of rings (see Section 7.3.8 [imap (plural)], page 285)

`preimage (plural)`
preimage under a ring map (see Section 7.3.21 [preimage (plural)], page 299)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 308)

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 "-" implicitely 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 283)

`freemodule`
: the free module of given rank (see Section 5.1.39 [freemodule], page 153)

`intersect`
: module intersection (see Section 7.3.9 [intersect (plural)], page 286)

`kbase` vector space basis of free module over the basering modulo the module of leading terms (see Section 7.3.10 [kbase (plural)], page 287)

`lead` initial module (see Section 5.1.66 [lead], page 172)

`lift` lift-matrix (see Section 7.3.11 [lift (plural)], page 288)

`liftstd` left Groebner basis and transformation matrix computation (see Section 7.3.12 [liftstd (plural)], page 289)

`modulo` represents $(h1+h2)/h1 \cong h2/(h1 \cap h2)$ (see Section 7.3.14 [modulo (plural)], page 291)

`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 291)

`ncols` number of columns (see Section 5.1.92 [ncols], page 190)

`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 295)

`nrows` number of rows (see Section 5.1.95 [nrows], page 191)

`oppose` creates an opposite module of a given module from the given ring into a basering (see Section 7.3.19 [oppose], page 296)

`print` nice print format (see Section 5.1.107 [print], page 200)

`prune` minimize the embedding into a free module (see Section 5.1.109 [prune], page 203)

`quotient` module quotient (see Section 7.3.22 [quotient (plural)], page 299)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 301)

`simplify` simplify a set of vectors (see Section 5.1.125 [simplify], page 216)

`size` number of non-zero generators (see Section 5.1.126 [size], page 217)

`std` left Groebner basis computation (see Section 7.3.26 [std (plural)], page 306)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 308)

`syz` computation of the first syzygy module (see Section 7.3.28 [syz (plural)], page 308)

`vdim` vector space dimension of free module over the basering modulo module of leading terms (see Section 7.3.30 [vdim (plural)], page 310)

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 273). 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 262).

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;
↦ //   characteristic : 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;
↦ //   characteristic : 0
↦ //   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 Lie bracket of two polinomials (see Section 7.3.2 [bracket], page 280)

`lead` leading term (see Section 5.1.66 [lead], page 172)

`leadcoef` coefficient of the leading term (see Section 5.1.67 [leadcoef], page 172)

`leadexp` the exponent vector of the leading monomial (see Section 5.1.68 [leadexp], page 173)

`leadmonom`
leading monomial (see Section 5.1.69 [leadmonom], page 173)

`oppose` creates an opposite polynomial of a given polynomial from the given ring into a basering (see Section 7.3.19 [oppose], page 296)

`reduce` left normal form with respect to a left Groebner basis (see Section 7.3.23 [reduce (plural)], page 301)

`simplify` normalize a polynomial (see Section 5.1.125 [simplify], page 216)

`size` number of monomials (see Section 5.1.126 [size], page 217)

`subst` substitute a ring variable (see Section 7.3.27 [subst (plural)], page 308)

`var` the indicated variable of the ring (see Section 5.1.146 [var], page 233)

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], page 309 command).

For a detailed description of the concept of rings and quotient rings see Section 3.3 [Rings and orderings], page 29.

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;
↦ //   characteristic : 0
↦ //   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 284)

`nvars` number of ring variables (see Section 5.1.96 [nvars], page 192)

`opposite` opposite ring (see Section 7.3.20 [opposite], page 297)

`setring` set a new basering (see Section 5.1.123 [setring], page 213)

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 279)

`minres` minimizes a free resolution (see Section 7.3.13 [minres (plural)], page 289)

`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 291)

`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 295)

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 29.

Note: PLURAL usually works with global orderings (see Section 7.1 [PLURAL], page 262) 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 292.

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;
↦ //   characteristic : 0
↦ //   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 5.1.6 [charstr], page 132)

`envelope` enveloping ring (see Section 7.3.6 [envelope], page 284)

`npars` number of ring parameters (see Section 5.1.93 [npars], page 190)

`nvars` number of ring variables (see Section 5.1.96 [nvars], page 192)

`opposite` opposite ring (see Section 7.3.20 [opposite], page 297)

`ordstr` monomial ordering of a ring (see Section 5.1.100 [ordstr], page 197)

`parstr` names of all ring parameters or the name of the n-th ring parameter (see Section 5.1.103 [parstr], page 198)

`qring` quotient ring (see Section 7.2.5 [qring (plural)], page 274)

`ringlist` decomposes a ring into a list of its components (see Section 7.3.24 [ringlist (plural)], page 302)

`setring` set a new basering (see Section 5.1.123 [setring], page 213)

`varstr` names of all ring variables or the name of the n-th ring variable (see Section 5.1.148 [varstr], page 234)

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 7.7 [Non-commutative libraries], page 320).

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 237, Section 5.3 [System variables], page 247).

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;
↦ //   characteristic : 0
↦ //   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 )`

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.

Example:

```
ring r=(0,Q),(x,y,z),Dp;
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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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
```

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.

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 4.3 [ideal], page 72; Section 4.11 [module], page 105; Section 5.1.133 [std], page 223; Section 5.1.149 [vdim], page 234.

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 4.3 [ideal], page 72; Section 5.1.71 [lift], page 174; Section 4.11 [module], page 105; Section 4.14 [poly], page 112; Section 4.20 [vector], page 124.

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 is possible. In these situations error messages will be reported.

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 263; Section 7.2.3 [module (plural)], page 269; Section 7.3.26 [std (plural)], page 306.

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 297.

```
LIB "ncalg.lib";
def A = makeUsl2();
setring A; A;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 296; Section 7.3.20 [opposite], page 297.

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 267 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 285; Section 7.2.2 [map (plural)], page 267; Section 7.2.5 [qring (plural)], page 274; Section 7.2.7 [ring (plural)], page 277.

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 267 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 285; Section 7.2.2 [map (plural)], page 267; Section 7.2.5 [qring (plural)], page 274; Section 7.2.7 [ring (plural)], page 277.

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 stucture if and only if the ideal is two-sided. `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 263; Section 7.2.3 [module (plural)], page 269; Section 7.3.30 [vdim (plural)], page 310.

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 a Groebner basis.
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);
print(matrix(i)); // print a compact presentation of i
↦ h+(-a),e
poly Z = 4*e*f+h^2-2*h; // a central element
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 263; Section 7.3.12 [liftstd (plural)], page 289; Section 7.2.3 [module (plural)], page 269.

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 263; Section 7.2.7 [ring (plural)], page 277; Section 7.3.26 [std (plural)], page 306.

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      3      3      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 291; Section 7.3.18 [nres (plural)], page 295.

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 308.

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(l[2])*transpose(l[1])));
↦ 0
print(matrix(M[3]));
↦ e,
↦ -f
// see the exactness at this point
size(ideal(transpose(l[3])*transpose(l[2])));
↦ 0
```

See Section 7.2.1 [ideal (plural)], page 263; Section 7.3.13 [minres (plural)], page 289; Section 7.2.3 [module (plural)], page 269; Section 7.3.18 [nres (plural)], page 295.

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 310.
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.`

Remark: At present, PLURAL does not check the non-degeneracy conditions (see Section 7.4.1 [G-algebras], page 310) 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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 7.7.8 [ncalg_lib], page 390; Section 7.7.10 [nctools_lib], page 412; Section 7.7.12 [qmatrix_lib], page 430.

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 292 but changes the basering.

Remark: This function is **depreciated** and should be substituted by `nc_algebra`, since it violates the general SINGULAR policy: only Section 4.18 [ring], page 118 and Section 5.1.123 [setring], page 213 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 292; Section 7.7.10 [nctools_lib], page 412.

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(l[2])*transpose(l[1])));
↦ 0
// see the exactness at this point:
size(ideal(transpose(l[3])*transpose(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 263; Section 7.3.13 [minres (plural)], page 289; Section 7.2.3 [module (plural)], page 269; Section 7.3.15 [mres (plural)], page 291.

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 297) 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;
```

```
↦ //   characteristic : 0
↦ //   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 284; Section 7.3.20 [opposite], page 297.

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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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 504; Section 7.3.6 [envelope], page 284; Section 7.3.19 [oppose], page 296.

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, it is implemented only for maps A -> B , where A is a commutative ring.

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-8
↦ _[3]=a-24
// hence, there are three simple characters for K
preimage(Usl2, Phi, Z);  // kernel pf phi
↦ _[1]=0
```

See Section 7.2.1 [ideal (plural)], page 263; Section 7.2.2 [map (plural)], page 267; Section 7.2.7 [ring (plural)], page 277.

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` modules 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), 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 263; Section 7.2.3 [module (plural)], page 269.

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 263; Section 7.2.3 [module (plural)], page 269; Section 7.3.26 [std (plural)], page 306.

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 310)
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 310)

Note: `After modifying a list aquired 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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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 277; Section 5.1.121 [ringlist], page 211.

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 449.

Example:

```
LIB "ncalg.lib";
LIB "nctools.lib";
↦ // ** redefining Gweights **
↦ // ** redefining Gweights **
↦ // ** redefining weightedRing **
↦ // ** redefining weightedRing **
↦ // ** redefining Cij **
↦ // ** redefining Ct **
↦ // ** redefining SimplMat **
↦ // ** redefining weightvector **
↦ // ** redefining ncRelations **
↦ // ** redefining ncRelations **
↦ // ** redefining findimAlgebra **
↦ // ** redefining findimAlgebra **
↦ // ** redefining isCentral **
↦ // ** redefining isCentral **
↦ // ** redefining UpOneMatrix **
↦ // ** redefining UpOneMatrix **
↦ // ** redefining ndcond **
↦ // ** redefining ndcond **
↦ // ** redefining Weyl **
↦ // ** redefining Weyl **
```

```
↦ // ** redefining makeHeisenberg **
↦ // ** redefining makeHeisenberg **
↦ // ** redefining superCommutative **
↦ // ** redefining superCommutative **
↦ // ** redefining SuperCommutative **
↦ // ** redefining SuperCommutative **
↦ // ** redefining ParseSCA **
↦ // ** redefining AltVarStart **
↦ // ** redefining AltVarStart **
↦ // ** redefining AltVarEnd **
↦ // ** redefining AltVarEnd **
↦ // ** redefining IsSCA **
↦ // ** redefining IsSCA **
↦ // ** redefining Exterior **
↦ // ** redefining Exterior **
↦ // ** redefining makeWeyl **
↦ // ** redefining makeWeyl **
↦ // ** redefining isNC **
↦ // ** redefining isNC **
↦ // ** redefining rightStd **
↦ // ** redefining rightStd **
↦ // ** redefining rightSyz **
↦ // ** redefining rightSyz **
↦ // ** redefining rightNF **
↦ // ** redefining rightNF **
↦ // ** redefining rightModulo **
↦ // ** redefining rightModulo **
↦ // ** redefining isCommutative **
↦ // ** redefining isCommutative **
↦ // ** redefining isWeyl **
↦ // ** redefining isWeyl **
↦ // ** redefining moduloSlim **
↦ // ** redefining moduloSlim **
↦ // ** redefining makeModElimRing **
↦ // ** redefining makeModElimRing **
def U = makeUsl(2);
// U is the U(sl_2) algebra
setring U;
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;
↦ //   characteristic : 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 5.1.98 [option], page 192; Section 7.3.26 [std (plural)], page 306.

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 312 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 5.1.98 [option], page 192(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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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 263; Section 7.2.7 [ring (plural)], page 277.

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
```

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;
option(redSB);
// we wish to have completely reduced bases:
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-1)*y,
↦ (Q)*z+(-Q+1),(Q-1)*z+(Q),(Q)*x+(-Q+1)*y,
↦ y+(-Q)*z,    x+(-Q),     (-Q)*x-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
// but the "commutative" syzygy property does not hold
size(ideal(matrix(K)*matrix(S)));
↦ 3
```

See also Section 7.2.1 [ideal (plural)], page 263; Section 7.3.13 [minres (plural)], page 289; Section 7.2.3 [module (plural)], page 269; Section 7.3.15 [mres (plural)], page 291; Section 7.3.18 [nres (plural)], page 295.

7.3.29 twostd

Syntax: `twostd(` ideal_expression`);`

Type: ideal

Purpose: returns a left Groebner basis of the two-sided ideal, generated by the input, treated as a set of two-sided generators. see Section 5.1.133 [std], page 223

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 Groebner basis, this is the same as the vector space dimension of the factor-module.

Note: In the non-commutative case, a ring modulo an ideal has a ring stucture 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 263; Section 7.3.10 [kbase (plural)], page 287; Section 7.3.26 [std (plural)], page 306.

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 314). 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 314).
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, $x_1 x_2$ is a monomial, while $x_2 x_1$ 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:

$A = K\langle x_1, \ldots, x_n \mid \{x_j x_i = c_{ij} \cdot x_i x_j + d_{ij}\}, 1 \le i < j \le n\rangle$, 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 $\forall i < j \;\; \mathrm{LM}(d_{ij}) < x_i x_j$,
- **non-degeneracy conditions**: $\forall\, 1 \le i < j < k \le n \;\; : \mathcal{NDC}_{ijk} = 0$, where

$$\mathcal{NDC}_{ijk} = c_{ik}c_{jk} \cdot d_{ij}x_k - x_k d_{ij} + c_{jk} \cdot x_j d_{ik} - c_{ij} \cdot d_{ik}x_j + d_{jk}x_i - c_{ij}c_{ik} \cdot x_i d_{jk}.$$

Note: Note that non-degeneracy conditions simply ensure associativity of multiplication.

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 277).
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 $i \ge j$ will be ignored).
2. $D = \{d_{ij}, i < j\}$, with polynomial entries d_{ij} from R (d_{ij} for $i \ge j$ will be ignored).

Call the initialization function `nc_algebra(C,D)` (see Section 7.3.16 [nc_algebra], page 292) with the data C and D.

At present, Plural does not check automatically whether the non-degeneracy conditions hold but it provides a procedure Section 7.7.10.3 [ndcond], page 414 from the library Section 7.7.10 [nctools_lib], page 412 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 507).

For a G–algebra A, we denote by ${}_A\langle g_1,\dots,g_s\rangle$ the left submodule of a free module A^r, generated by elements $\{g_1,\dots,g_s\}\subset A^r$.

Let $<$ be a fixed monomial well-ordering on the G–algebra A with the Poincaré-Birkhoff-Witt (PBW) basis $\{x^\alpha = x_1^{a_1}x_2^{a_2}\dots x_n^{a_n}\}$. For a given free module A^r with the basis $\{e_1,\dots,e_r\}$, $<$ denotes also a fixed module ordering on the set of monomials $\{x^\alpha e_i \mid \alpha\in\mathbf{N}^n, 1\le i\le r\}$.

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, $L(S)=\oplus\{Kx^\alpha e_i \mid \exists s\in S, \mathrm{LM(s)} = x^\alpha e_i\}$. 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 $\beta_j\le\alpha_j$, $1\le j\le n$.

Remark: In general non-commutative algorithms are working with well-orderings only (see Section 7.1 [PLURAL], page 262, Section B.2 [Monomial orderings], page 502 and Section 3.3.3 [Term orderings], page 33), unless we deal with grade commutative algebras.

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 $\mathrm{NF}: A^r\times\{G\mid G$ a (left) Groebner basis$\}\to A^r, (f|G)\mapsto\mathrm{NF}(f|G)$, 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) $\mathrm{NF}(0|G)=0$,

(ii) if $\mathrm{NF}(f|G)\neq 0$ then $\mathrm{LM}(g)$ does not divide $\mathrm{LM}(\mathrm{NF}(f|G))$ for all $g\in G$,

(iii) $f-\mathrm{NF}(f|G)\in{}_A\langle G\rangle$.

$\mathrm{NF}(f|G)$ 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 262), PLURAL works with left normal form only.

Left ideal membership

For a left Groebner basis G of I the following holds: $f\in I$ if and only if the left normal form $\mathrm{NF}(f|G)=0$.

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 262), PLURAL works with left syzygies only (by `syz` we understand a left syzygy). 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(l[3])*transpose(l[2]); // 2nd term
size(ideal(TST));
↦ 0
TST = transpose(l[2])*transpose(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 `http://www.mathematik.uni-kl.de/~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
- [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 co-operation of L. W. Small. Graduate Studies in Mathematics. 30. Providence, RI: American Mathematical Society (AMS)., 2001

Descriptions of algorithms and problems

- Havlicek, M. and Klimyk, A. and Posta, S. Central elements of the algebras $U'_q(\mathrm{so}_m)$ and $U'_q(\mathrm{iso}_m)$. arXiv. math. QA/9911130, (1999)
- 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.
- N. Iorgov. On the Center of q-Deformed Algebra $U'_q(\mathrm{so}_3)$ Related to Quantum Gravity at q a Root of 1. In Proceedings of IV Int. Conf. "Symmetry in Nonlinear Mathematical Physics",(2001) Kyiv, Ukraine
- A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. J. Symbolic Computation, 9(1):1-26, 1990.
- Levandovskyy, V. On Gröbner bases for non-commutative G-algebras. In Kredel, H. and Seiler, W.K., editor, Proceedings of the 8th Rhine Workshop on Computer Algebra, 2002.
- [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 non-commutative 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. Available online at `http://www.mathematik.uni-kl.de/~zca/Reports_on_ca/index.html`.
- 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 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 grade 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_2 -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:

$$K\langle x_1,\dots,x_n;y_1,\dots,y_m \mid y_j * y_i = -y_iy_j, j>i\rangle / \langle y_1^2,\dots,y_m^2\rangle .$$

Distinctive features

Graded commutative algebras have zero divisors iff $m > 0 : y_i * y_i = 0$.

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

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;
↦ //   characteristic : 0
↦ //   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;
// 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;
↦ // ** redefining S **
↦ //   characteristic : 0
↦ //   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]=aW+z+bxQ
↦ _[2]=xz
↦ _[3]=yz-bxyQ
↦ _[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]=a+yQ
↦ _[2]=zQ+b
↦ _[3]=bx
```

```
↦ _[4]=by
↦ _[5]=bz
↦ _[6]=b2
↦ _[7]=yQW-z-bxQ
```

See example of `superCommutative` from the library `nctools.lib`.

Reference: Ph.D thesis by Oleksandr Motsak (2010).

7.6 LETTERPLACE

This section describes mathematical notions and definitions used in the experimental LETTERPLACE extension of SINGULAR.

For further details, please, refer to the paper [LL]: 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.

All algebras are assumed to be associative K -algebras for some field K .

7.6.1 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 a tensor algebra $T(V)$ of V . It is an infinite dimensional K -vector space, where one can take as a basis the elements of the free monoid < x_1 ,. . . , x_n > together with 1. 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 Noetherian for $n > 1$ (hence, a Groebner basis of a finitely generated ideal might be infinite). The free associative algebra can be regarded as a graded algebra in a natural way.

Any finitely presented associative algebra is isomorphic to a quotient of K < x_1 ,. . . , x_n > modulo a two-sided ideal.

7.6.2 Groebner bases for two-sided ideals in free associative algebras

We call a total ordering $<$ on the free monoid $X := \langle x_1, \ldots, x_n \rangle$ a **monomial ordering if the following conditions hold:**

$<$ is a well-ordering on X, that is $1 < x \ \forall x \in X$,

$\forall p, q, s, t \in X$, if $s < t$, then $p \cdot s \cdot q < p \cdot t \cdot q$,

$\forall p, q, s, t \in X$, if $s = p \cdot t \cdot q$ and $s \neq t$, then $t < s$.

Hence the notions of a leading monomial and coefficients transfer to this situation.

We say that a monomial v divides monomial w , if there exist monomials $p, s \in X$, such that $w = p \cdot v \cdot s$.

For a subset $G \subset T := K\langle x_1, \ldots, x_n \rangle$, define a **leading ideal of G to be the two-sided ideal** $L(G) = {}_T\langle \ \{lm(g) \mid g \in G \setminus \{0\}\} \ \rangle_T \subseteq T$.

Let $<$ be a fixed monomial ordering on T. We say that a subset $G \subset I$ is a (two-sided) Groebner basis for the ideal I with respect to $<$, if $L(G) = L(I)$. That is $\forall f \in I$ there exists $g \in G$, such that $lm(g)$ divides $lm(f)$.

7.6.3 Letterplace correspondence

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 monoid of natural numbers, starting with 0 which cannot be used as a place.

Note, that the latter letterplace algebra $K[X\times N]$ is an infinitely generated commutative polynomial algebra. Since $K <$

x_1 ,. . . , x_n

$>$ is not Noetherian, it is common to perform the computations up to a given degree. In that case the truncated letterplace algebra is (a) finitely generated (commutative ring). In [LL] a natural shifting on letterplace polynomials was introduced and used. Indeed, there is 1-to-1 correspondence between graded two-sided ideals of a free algebra and so-called letterplace ideals in the letterplace algebra, see [LL] for details. All the computations take place in the latter algebra. A letterplace monomial of length m is a monomial of a letterplace algebra, such that its m places are exactly 1,2,. . . , m . That is, such monomials are multilinear with respect to places. A letterplace ideal is generated by letterplace polynomials, which start from the place 1 (i.e. elements from the special vector space V) subject to two kind of operations: the K-algebra operations of the letterplace algebra and simultaneous shifting of places by any natural number n.

7.6.4 Example of use of Letterplace

The input monomials must be given in a letterplace notation. We recommend first to define a commutative ring $K[X]$ in Singular and equip it with a degree well-ordering. Then, decide what should be the degree bound d and run the procedure `makeLetterplaceRing(d)` from the library `freegb.lib`. This procedure creates a letterplace algebra with an ordering, induced from the given commutative ring $K[X]$. In this algebra, define an ideal `I` as a list of polynomials in the letterplace encoding and run the procedure `system("freegb",I,d,n)`, where n is the number of variables of the original commutative ring. The output is given in the letterplace encoding as well.

Note that one can also use `freeGBasis` from `freegb.lib` in order to use a different encoding for polynomials in free algebra.

We illustrate this approach with the following example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
int d =4; // degree bound
def R = makeLetterplaceRing(d);
setring R;
ideal I = x(1)*y(2) + y(1)*z(2), x(1)*x(2) + x(1)*y(2) - y(1)*x(2) - y(1)*y(2);
option(redSB); option(redTail);
ideal J = system("freegb",I,d,nvars(r));
J;
↦ J[1]=x(1)*y(2)+y(1)*z(2)
↦ J[2]=x(1)*x(2)-y(1)*x(2)-y(1)*y(2)-y(1)*z(2)
↦ J[3]=y(1)*y(2)*y(3)-y(1)*y(2)*z(3)+y(1)*z(2)*y(3)-y(1)*z(2)*z(3)
↦ J[4]=y(1)*y(2)*x(3)+y(1)*y(2)*z(3)+y(1)*z(2)*x(3)+y(1)*z(2)*z(3)
↦ J[5]=y(1)*z(2)*y(3)*y(4)-y(1)*z(2)*y(3)*z(4)+y(1)*z(2)*z(3)*y(4)-y(1)*z(2\
  )*z(3)*z(4)
↦ J[6]=y(1)*z(2)*y(3)*x(4)+y(1)*z(2)*y(3)*z(4)+y(1)*z(2)*z(3)*x(4)+y(1)*z(2\
  )*z(3)*z(4)
↦ J[7]=y(1)*y(2)*z(3)*y(4)-y(1)*y(2)*z(3)*z(4)+y(1)*z(2)*z(3)*y(4)-y(1)*z(2\
```

```
   )*z(3)*z(4)
↦ J[8]=y(1)*y(2)*z(3)*x(4)+y(1)*y(2)*z(3)*z(4)+y(1)*z(2)*z(3)*x(4)+y(1)*z(2\
   )*z(3)*z(4)
// ----------------------------------
lp2lstr(J,r); // export an object called @code{@LN} to the ring r
setring r;  // change to the ring r
lst2str(@LN,1); // output the string presentation
↦ [1]:
↦    x*y+y*z
↦ [2]:
↦    x*x-y*x-y*y-y*z
↦ [3]:
↦    y*y*y-y*y*z+y*z*y-y*z*z
↦ [4]:
↦    y*y*x+y*y*z+y*z*x+y*z*z
↦ [5]:
↦    y*z*y*y-y*z*y*z+y*z*z*y-y*z*z*z
↦ [6]:
↦    y*z*y*x+y*z*y*z+y*z*z*x+y*z*z*z
↦ [7]:
↦    y*y*z*y-y*y*z*z+y*z*z*y-y*z*z*z
↦ [8]:
↦    y*y*z*x+y*y*z*z+y*z*z*x+y*z*z*z
```

It is possible to convert the letterplace presentation of an ideal to a list of strings with the help of procedures `lp2lstr` and `lst2str` from the library `freegb.lib` (see see Section 7.7.5 [freegb_lib], page 375). This is shown in the second part of the example above.

There are various conversion routines in the library `freegb.lib` (see see Section 7.7.5 [freegb_lib], page 375). We work further on implementing more algorithms for non-commutative ideals and modules over free associative algebra.

7.6.5 Release notes of Letterplace

With this functionality it is possible to compute two-sided Groebner basis of a graded two-sided ideal (that is, generated by homogeneous polynomials) in a free associative algebra up to a given degree.

Restrictions of the Letterplace package:

- At the moment we provide stable implementation for the homogeneous input only, computations with inhomogeneous ideals are under development. (There are no automatic checks.)
- Since free algebra is not Noetherian, one has to define an explicit degree bound, up to which a partial Groebner basis will be computed.
- the options `option(redSB); option(redTail);` must be always activated
- we advise to run the computations with the options `option(prot);option(mem);` in order to see the activity journal as well as the memory usage

7.7 Non-commutative libraries

Plural comes with a set of standard libraries. Their content 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.

7.7.1 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

Theory: 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).

Main procedures: Auxiliary procedures: See also: Section 7.7.3 [dmod_lib], page 346; Section 7.7.4 [dmodapp_lib], page 364; Section D.5.7 [gmssing_lib], page 919.

7.7.1.1 bfct

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]=-7/6
↦    _[3]=-1
↦ [2]:
↦    1,1,1
intvec v = 3,2;
bfct(f,1,0,v);
↦ [1]:
↦    _[1]=-1
↦    _[2]=-7/6
↦    _[3]=-5/6
↦ [2]:
↦    1,1,1
```

7.7.1.2 bfctSyz

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]=-7/6
↦    _[3]=-1
↦ [2]:
↦    1,1,1
intvec v = 3,2;
bfctSyz(f,0,1,1,0,v);
↦ [1]:
↦    _[1]=-1
↦    _[2]=-7/6
↦    _[3]=-5/6
↦ [2]:
↦    1,1,1
```

7.7.1.3 bfctAnn

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]=-1
↦    _[2]=-5/6
↦    _[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]=-17/20
↦    _[2]=-21/20
↦    _[3]=-9/10
↦    _[4]=-23/20
↦    _[5]=-7/10
↦    _[6]=-11/20
↦    _[7]=-9/20
↦    _[8]=-19/20
↦    _[9]=-13/20
↦    _[10]=-13/10
↦    _[11]=-27/20
↦    _[12]=-1
↦    _[13]=-11/10
↦ [2]:
↦    1,1,1,1,1,1,1,1,1,1,1,1,1
```

7.7.1.4 bfctOneGB

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]=-7/6
↦    _[3]=-1
↦ [2]:
↦    1,1,1
bfctOneGB(f,1,1);
↦ [1]:
↦    _[1]=-1
↦    _[2]=-7/6
↦    _[3]=-5/6
↦ [2]:
↦    1,1,1
```

7.7.1.5 bfctIdeal

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]=-4/3
↦    _[3]=-2/3
↦ [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
```

7.7.1.6 pIntersect

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.1.7 pIntersectSyz

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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)
```

7.7.1.8 linReduce

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.1.9 linReduceIdeal

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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]) - M*l[2];
↦ _[1,1]=0
↦ _[1,2]=0
↦ _[1,3]=0
↦ _[1,4]=0
```

7.7.1.10 linSyzSolve

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.1.11 allPositive

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.1.12 scalarProd

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.1.13 vec2poly

Procedure from library `bfun.lib` (see Section 7.7.1 [bfun_lib], page 321).

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
```

7.7.2 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:

7.7.2.1 centralizeSet

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.7 [centralizer], page 336; Section 7.7.2.2 [centralizerVS], page 333; Section 7.7.2.11 [inCentralizer], page 339.

7.7.2.2 centralizerVS

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.4 [centerVS], page 334; Section 7.7.2.7 [centralizer], page 336; Section 7.7.2.11 [inCentralizer], page 339.

7.7.2.3 centralizerRed

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.5 [centerRed], page 334; Section 7.7.2.7 [centralizer], page 336; Section 7.7.2.2 [centralizerVS], page 333; Section 7.7.2.11 [inCentralizer], page 339.

7.7.2.4 centerVS

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.6 [center], page 335; Section 7.7.2.2 [centralizerVS], page 333; Section 7.7.2.10 [inCenter], page 338.

7.7.2.5 centerRed

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.6 [center], page 335; Section 7.7.2.4 [centerVS], page 334; Section 7.7.2.3 [centralizerRed], page 333; Section 7.7.2.10 [inCenter], page 338.

7.7.2.6 center

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 ZZ = center(-1, 1);
ZZ;
↦ ZZ[1]=t
inCenter(ZZ); // check the result
↦ 1
```

See also: Section 7.7.2.7 [centralizer], page 336; Section 7.7.2.10 [inCenter], page 338.

7.7.2.7 centralizer

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.6 [center], page 335; Section 7.7.2.11 [inCentralizer], page 339.

7.7.2.8 sa_reduce

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.9 [sa_poly_reduce], page 338.

7.7.2.9 sa_poly_reduce

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.8 [sa_reduce], page 337.

7.7.2.10 inCenter

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.2.11 inCentralizer

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.2.12 isCartan

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.2.13 applyAdF

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.14 [linearMapKernel], page 341.

7.7.2.14 linearMapKernel

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.13 [applyAdF], page 340; Section 7.7.2.14 [linearMapKernel], page 341.

7.7.2.15 linearCombinations

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.13 [applyAdF], page 340; Section 7.7.2.14 [linearMapKernel], page 341.

7.7.2.16 variablesStandard

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.17 [variablesSorted], page 343.

7.7.2.17 variablesSorted

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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 7.7.2.16 [variablesStandard], page 343.

7.7.2.18 PBW_eqDeg

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.2.19 PBW_maxDeg

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.2.20 PBW_maxMonom

Procedure from library `central.lib` (see Section 7.7.2 [central_lib], page 332).

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
```

7.7.3 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

Theory: 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

Main procedures: Auxiliary procedures: See also: Section 7.7.1 [bfun_lib], page 321; Section 7.7.4 [dmodapp_lib], page 364; Section D.5.7 [gmssing_lib], page 919.

7.7.3.1 annfs

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-7/6
↦    _[2]=-5/6
↦    _[3]=-4/3
↦    _[4]=-1
```

```
↦     _[5]=-5/3
↦  [2]:
↦     1,1,1,1,1
```

7.7.3.2 annfspecial

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

Usage: annfspecial(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.

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;
annfspecial(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]=108*y^2*Dy^3+32*y*Dx^3-216*y*Dy^2+231*Dy
annfspecial(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+243*y*Dy^2+429*Dy
annfspecial(LD,F,-1,1);   // exceptional 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
```

7.7.3.3 Sannfs

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.4 Sannfslog

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.5 bernsteinBM

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-7/3
↦    _[2]=-5/3
↦    _[3]=-7/6
↦    _[4]=-1
↦    _[5]=-11/6
↦    _[6]=-2
↦    _[7]=-3/2
↦ [2]:
↦    1,1,1,1,1,1,1
```

7.7.3.6 bernsteinLift

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-1
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-2
↦ [2]:
↦    2,1,1,1
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
```

7.7.3.7 operatorBM

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-1
↦    _[2]=-4/3
↦    _[3]=-5/3
↦    _[4]=-2
↦ [2]:
↦    2,1,1,1
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
```

7.7.3.8 operatorModulo

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.9 annfsParamBM

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.10 annfsBMI

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

Usage: annfsBMI(F [,eng]); F an ideal, eng an optional int

Return: ring

Purpose: compute the D-module structure of basering[1/f]*f^s where
f = F[1]*..*F[P], according to the algorithm by Briancon and Maisonobe.

Note: activate the output ring with the `setring` command. In this ring,
- the ideal LD is the needed D-mod structure,
- the list BS is the Bernstein 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 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;
def A = annfsBMI(F);
setring A;
LD;
↦ 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;
↦ [1]:
↦    _[1]=s(1)+1
↦    _[2]=s(2)+1
↦    _[3]=s(1)+s(2)+s(3)+4
↦    _[4]=s(1)+s(2)+s(3)+2
↦    _[5]=s(1)+s(2)+s(3)+3
↦    _[6]=s(3)+1
↦ [2]:
↦    1,1,1,1,1,1
```

7.7.3.11 checkRoot

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.12 SannfsBFCT

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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;
↦ //   characteristic : 0
↦ //   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^6-4244944536107/25600000000*s^\
   5-3066298289417/25600000000*s^4-2787777479229/51200000000*s^3-19980507461\
   787/1280000000000*s^2-663659243177931/256000000000000*s-48839201079669/25\
   6000000000000
```

7.7.3.13 annfs0

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-1
↦     _[2]=-4/3
↦     _[3]=-5/3
↦     _[4]=-2
↦  [2]:
↦     2,1,1,1
```

7.7.3.14 annfs2

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-1
↦     _[2]=-2
↦     _[3]=-5/3
```

```
↦     _[4]=-4/3
↦  [2]:
↦     2,1,1,1
```

7.7.3.15 annfsRB

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=-1
↦     _[2]=-2
↦     _[3]=-5/3
↦     _[4]=-4/3
↦ [2]:
↦     2,1,1,1
```

7.7.3.16 checkFactor

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.17 arrange

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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]=x+z
```

```
↦ _[3]=x+y
↦ _[4]=x+y+z
↦ _[5]=y
↦ _[6]=y+z
↦ _[7]=z
```

7.7.3.18 reiffen

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.19 isHolonomic

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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 stays 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
```

7.7.3.20 convloc

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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;
↦ //   characteristic : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering Ds
↦ //                  : names    x y
↦ //        block   2 : ordering ds
↦ //                  : names    z
↦ //        block   3 : ordering C
```

7.7.3.21 minIntRoot

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.3.22 varNum

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

Usage: varNum(s); string s

Return: int

Purpose: returns the number of the variable with the name s
among the variables of basering or 0 if there is no such variable

Example:

```
LIB "dmod.lib";
ring X = 0,(x,y1,t,z(0),z,tTa),dp;
varNum("z");
↦ 5
varNum("t");
↦ 3
varNum("xyz");
↦ 0
```

7.7.3.23 isRational

Procedure from library `dmod.lib` (see Section 7.7.3 [dmod_lib], page 346).

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
```

7.7.4 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

Guide: 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:
- localization of a holonomic module D/I with respect to a mult. closed set of all powers of a given polynomial F from R. Our aim is to compute an ideal L in D, such that D/L is a presentation of a localized module. Such L always exists, since such localizations are known to be holonomic and thus cyclic modules. 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.

- initial form and initial ideals in Weyl algebras with respect to a given weight vector can be computed with inForm, initialMalgrange, initialIdealW.

- 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
(ONW) Oaku, Takayama, Walther 'A Localization Algorithm for D-modules', 2000

Main procedures: Auxiliary procedures: See also: Section 7.7.1 [bfun_lib], page 321; Section 7.7.3 [dmod_lib], page 346; Section D.5.7 [gmssing_lib], page 919.

7.7.4.1 annPoly

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: annPoly(f); f a poly

Return: ring

Purpose: compute the complete annihilator ideal of f in the Weyl algebra D

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
```

See also: Section 7.7.4.2 [annRat], page 365.

7.7.4.2 annRat

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: annRat(g,f); f, g polynomials

Return: ring

Purpose: compute the annihilator of the rational function g/f in the Weyl algebra D

Note: activate the output ring with the `setring` command.
In the output ring, the ideal LD (in Groebner basis) is the annihilator.
The algorithm uses the computation of ann f^{-1} via D-modules.

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 7.7.4.1 [annPoly], page 364.

7.7.4.3 DLoc

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: DLoc(I, F); I an ideal, F a poly

Return: nothing (exports objects instead)

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 basering, the following objects are exported:
the ideal LD0 (in Groebner basis) is the presentation of the localization
the list BS contains 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
DLoc(I, x2-y3); // exports LD0 and BS
LD0; // localized module (R/I)_f is isomorphic to R/LD0
↦ 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
```

7.7.4.4 SDLoc

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: SDLoc(I, F); I an ideal, F a poly

Return: ring

Purpose: compute a generic presentation of the localization of D/I w.r.t. f^s

Assume: the basering D is a Weyl algebra

Note: activate this ring with the `setring` command. In this ring,
the ideal LD (in 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
```

7.7.4.5 DLoc0

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: DLoc0(I, F); I an ideal, F a poly

Return: ring

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 this ring with the `setring` command. In this ring,
the ideal LD0 (in 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
```

7.7.4.6 initialMalgrange

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

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 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
```

7.7.4.7 initialIdealW

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

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 w.r.t. 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 consist of 2n strictly positive entries, w is used for weighted homogenization, 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 "dmodapp.lib";
ring @D = 0,(x,Dx),dp;
def D = Weyl();
setring D;
intvec u = -1; intvec v = 2;
ideal I = x^2*Dx^2,x*Dx^4;
ideal J = initialIdealW(I,u,v); J;
↦ J[1]=Dx^2
```

7.7.4.8 inForm

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: inForm(I,w); I ideal, w intvec

Return: the initial form of I w.r.t. the weight vector w

Purpose: computes the initial form of an ideal w.r.t. a given weight vector

Note: the size of the weight vector must be equal to the number of variables
of the basering.

Example:

```
LIB "dmodapp.lib";
ring @D = 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
```

7.7.4.9 isFsat

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: isFsat(I, F); I an ideal, F a poly

Return: int

Purpose: check whether the ideal I is F-saturated

Note: 1 is returned if I is F-saturated, otherwise 0 is returned.
we check indeed that Ker(D –F–> D/I) is (0)

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
```

7.7.4.10 bFactor

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: bFactor(f); f poly

Return: list

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.

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
```

7.7.4.11 appelF1

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: appelF1();

Return: ring (and exports an ideal into it)

Purpose: define the ideal in a parametric Weyl algebra,
which annihilates Appel F1 hypergeometric function

Note: the ideal called IAppel1 is exported to the output ring

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
```

7.7.4.12 appelF2

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: appelF2();

Return: ring (and exports an ideal into it)

Purpose: define the ideal in a parametric Weyl algebra,
which annihilates Appel F2 hypergeometric function

Note: the ideal called IAppel2 is exported to the output ring

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
```

7.7.4.13 appelF4

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: appelF4();

Return: ring (and exports an ideal into it)

Purpose: define the ideal in a parametric Weyl algebra,
which annihilates Appel F4 hypergeometric function

Note: the ideal called IAppel4 is exported to the output ring

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)
```

7.7.4.14 engine

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: engine(I,i); I ideal/module/matrix, i an int

Return: the same type as I

Purpose: compute the Groebner basis of I with the algorithm, chosen via i

Note: By default and if i=0, slimgb is used; otherwise std does the job.

Example:

```
LIB "dmodapp.lib";
ring r = 0,(x,y),Dp;
ideal I  = y*(x3-y2),x*(x3-y2);
engine(I,0); // uses slimgb
↦ _[1]=x3y-y3
↦ _[2]=x4-xy2
engine(I,1); // uses std
↦ _[1]=x3y-y3
↦ _[2]=x4-xy2
```

7.7.4.15 poly2list

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: poly2list(f); f a poly

Return: list of exponents and corresponding terms of f

Purpose: convert a polynomial to a list of exponents and corresponding terms

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+x*y2;
poly2list(F);
↦ [1]:
↦    [1]:
↦       2,1
↦    [2]:
↦       x2y
↦ [2]:
↦    [1]:
↦       1,2
↦    [2]:
↦       xy2
```

7.7.4.16 fl2poly

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

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,-5/3,-2;
intvec mI = 2,1,1,1;
list BS = I,mI;
poly p = fl2poly(BS,"s");
p;
↦ s5+7s4+173/9s3+233/9s2+154/9s+40/9
factorize(p,2);
↦ [1]:
↦    _[1]=s+1
↦    _[2]=3s+4
↦    _[3]=3s+5
↦    _[4]=s+2
↦ [2]:
↦    2,1,1,1
```

7.7.4.17 insertGenerator

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: insertGenerator(id,p[,k]); id an ideal/module, p a poly/vector, k an optional int

Return: same as id

Purpose: inserts p into the first argument at k-th index 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.

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;
insertGenerator(M,[x^3,y^2,z],2);
↦ _[1]=x2*gen(1)
↦ _[2]=x3*gen(1)+y2*gen(2)+z*gen(3)
↦ _[3]=z4*gen(1)
```

7.7.4.18 deleteGenerator

Procedure from library `dmodapp.lib` (see Section 7.7.4 [dmodapp_lib], page 364).

Usage: deleteGenerator(id,k); id an ideal/module, k an int

Return: same 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];;
deleteGenerator(M,2);
↦ _[1]=x*gen(1)+y*gen(2)+z*gen(3)
↦ _[2]=x3*gen(1)+y3*gen(2)+z3*gen(3)
```

7.7.5 freegb_lib

Library: freegb.lib

Purpose: Compute two-sided Groebner bases in free algebras via letterplace

Author: Viktor Levandovskyy, levandov@math.rwth-aachen.de

Theory: See chapter 'LETTERPLACE' in the @sc{Singular} Manual.

Procedures: Auxiliary procedures: See also: Section 7.6 [LETTERPLACE], page 318.

7.7.5.1 makeLetterplaceRing

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: makeLetterplaceRing(d [,h]); d an integer, h an optional integer

Return: ring

Purpose: creates a ring with the ordering, used in letterplace computations

Note: if h is given an nonzero, the pure homogeneous letterplace block ordering will be used.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),(dp(1),dp(2));
def A = makeLetterplaceRing(2);
setring A;  A;
↦ //   characteristic : 0
↦ //   number of vars : 6
↦ //        block   1 : ordering a
↦ //                  : names    x(1) y(1) z(1) x(2) y(2) z(2)
↦ //                  : weights     1    1    1    1    1    1
↦ //        block   2 : ordering dp
↦ //                  : names    x(1)
↦ //        block   3 : ordering dp
↦ //                  : names    y(1) z(1)
↦ //        block   4 : ordering dp
↦ //                  : names    x(2)
↦ //        block   5 : ordering dp
↦ //                  : names    y(2) z(2)
↦ //        block   6 : ordering C
attrib(A,"isLetterplaceRing");
↦ 1
attrib(A,"uptodeg");  // degree bound
↦ 2
attrib(A,"lV"); // number of variables in the main block
↦ 3
setring r; def B = makeLetterplaceRing(2,1); // to compare:
setring B;  B;
↦ //   characteristic : 0
↦ //   number of vars : 6
↦ //        block   1 : ordering dp
↦ //                  : names    x(1)
↦ //        block   2 : ordering dp
↦ //                  : names    y(1) z(1)
↦ //        block   3 : ordering dp
↦ //                  : names    x(2)
↦ //        block   4 : ordering dp
↦ //                  : names    y(2) z(2)
↦ //        block   5 : ordering C
```

7.7.5.2 freeGBasis

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: freeGBasis(L, d); L a list of modules, d an integer

Return: ring

Assume: L has a special form. Namely, it is a list of modules, where each generator of every module stands for a monomial times coefficient in free algebra. In such a vector generator, the 1st entry is a nonzero coefficient from the ground field and each next entry hosts a variable from the basering.

Purpose: compute the two-sided Groebner basis of an ideal, encoded by L in the free associative algebra, up to degree d

Note: Apply `lst2str` to the output in order to obtain a better readable presentation

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),(dp(1),dp(2));
module M = [-1,x,y],[-7,y,y],[3,x,x]; // stands for free poly -xy - 7yy - 3xx
module N = [1,x,y,x],[-1,y,x,y]; // stands for free poly xyx - yxy
list L; L[1] = M; L[2] = N; // list of modules stands for an ideal in free algebra
lst2str(L); // list to string conversion of input polynomials
↦ [1]:
↦    -xy-7yy+3xx
↦ [2]:
↦    xyx-yxy
def U = freeGBasis(L,5); // 5 is the degree bound
lst2str(U);
↦ [1]:
↦    yyyyy
↦ [2]:
↦    22803yyyx+19307yyyy
↦ [3]:
↦    1933yyxy+2751yyyx+161yyyy
↦ [4]:
↦    22xyy-3yxy-21yyx+7yyy
↦ [5]:
↦    3xyx-22xyy+21yyx-7yyy
↦ [6]:
↦    3xx-xy-7yy
```

7.7.5.3 setLetterplaceAttributes

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

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' = true, 'uptodeg' = d, 'lV' = b, 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;
```

```
attrib(R,"isLetterplaceRing");
↦ 1
lieBracket(x(1),y(1),2);
↦ -y(1)*x(2)*x(3)*x(4)+3*x(1)*y(2)*x(3)*x(4)-3*x(1)*x(2)*y(3)*x(4)+x(1)*x(2\
  )*x(3)*y(4)
```

7.7.5.4 lpMult

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: lpMult(f,g); f,g letterplace polynomials

Return: poly

Assume: basering has a letterplace ring structure

Purpose: compute the letterplace form of f*g

Example:

```
LIB "freegb.lib";
// define a ring in letterplace form as follows:
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); // supply R with letterplace structure
setring R;
poly a = x(1)*y(2); poly b = y(1);
lpMult(b,a);
↦ y(1)*x(2)*y(3)
lpMult(a,b);
↦ x(1)*y(2)*y(3)
```

7.7.5.5 shiftPoly

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: shiftPoly(p,i); p letterplace poly, i int

Return: poly

Assume: basering has letterplace ring structure

Purpose: compute the i-th shift of letterplace polynomial p

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),dp;
int uptodeg = 5; int lV = 3;
def R = makeLetterplaceRing(uptodeg);
setring R;
poly f = x(1)*z(2)*y(3) - 2*z(1)*y(2) + 3*x(1);
shiftPoly(f,1);
↦ x(2)*z(3)*y(4)-2*z(2)*y(3)+3*x(2)
shiftPoly(f,2);
↦ x(3)*z(4)*y(5)-2*z(3)*y(4)+3*x(3)
```

7.7.5.6 lpPower

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: lpPower(f,n); f letterplace polynomial, int n

Return: poly

Assume: basering has a letterplace ring structure

Purpose: compute the letterplace form of f^n

Example:

```
LIB "freegb.lib";
// define a ring in letterplace form as follows:
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); // supply R with letterplace structure
setring R;
poly a = x(1)*y(2); poly b = y(1);
lpPower(a,2);
↦ x(1)*y(2)*x(3)*y(4)
lpPower(b,4);
↦ y(1)*y(2)*y(3)*y(4)
```

7.7.5.7 lp2lstr

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: lp2lstr(K,s); K an ideal, s a ring name

Return: nothing (exports object `@LN` into the ring named s)

Assume: basering has a letterplace ring structure

Purpose: converts letterplace ideal to list of modules

Note: useful as preprocessing to 'lst2str'

Example:

```
LIB "freegb.lib";
intmat A[2][2] = 2, -1, -1, 2; // sl_3 == A_2
ring r = 0,(f1,f2),dp;
def R = makeLetterplaceRing(3);
setring R;
ideal I = serreRelations(A,1);
lp2lstr(I,r);
setring r;
lst2str(@LN,1);
↦ [1]:
↦    f1*f2*f2-2*f2*f1*f2+f2*f2*f1
↦ [2]:
↦    f1*f1*f2-2*f1*f2*f1+f2*f1*f1
```

7.7.5.8 lst2str

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: lst2str(L[,n]); L a list of modules, n an optional integer

Return: list (of strings)

Purpose: convert a list (of modules) into polynomials in free algebra

Note: if an optional integer is not 0, stars signs are used in multiplication

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),(dp(1),dp(2));
module M = [-1,x,y],[-7,y,y],[3,x,x];
module N = [1,x,y,x,y],[-2,y,x,y,x],[6,x,y,y,x,y];
list L; L[1] = M; L[2] = N;
lst2str(L);
↦ [1]:
↦    -xy-7yy+3xx
↦ [2]:
↦    xyxy-2yxyx+6xyyxy
lst2str(L[1],1);
↦ [1]:
↦    -x*y-7*y*y+3*x*x
```

7.7.5.9 mod2str

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: mod2str(M[,n]); M a module, n an optional integer

Return: string

Purpose: convert a module into a polynomial in free algebra

Note: if an optional integer is not 0, stars signs are used in multiplication

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y,z),(dp);
module M = [1,x,y,x,y],[-2,y,x,y,x],[6,x,y,y,x,y];
mod2str(M);
↦ xyxy-2yxyx+6xyyxy
mod2str(M,1);
↦ x*y*x*y-2*y*x*y*x+6*x*y*y*x*y
```

7.7.5.10 vct2str

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: vct2str(v[,n]); v a vector, n an optional integer

Return: string

Purpose: convert a vector into a word in free algebra

Note: if an optional integer is not 0, stars signs are used in multiplication

Example:

```
LIB "freegb.lib";
ring r = (0,a),(x,y3,z(1)),dp;
vector v = [-7,x,y3^4,x2,z(1)^3];
vct2str(v);
```

```
↦ -7xy3y3y3y3xxz(1)z(1)z(1)
vct2str(v,1);
↦ -7*x*y3*y3*y3*y3*x*x*z(1)*z(1)*z(1)
vector w = [-7a^5+6a,x,y3,y3,x,z(1),z(1)];
vct2str(w);
↦ (-16807*a^5+6*a)xy3y3xz(1)z(1)
vct2str(w,1);
↦ (-16807*a^5+6*a)*x*y3*y3*x*z(1)*z(1)
```

7.7.5.11 lieBracket

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

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(1),y(1),x(2),y(2),x(3),y(3),x(4),y(4)),dp;
def R = setLetterplaceAttributes(r,4,2); // supply R with letterplace structure
setring R;
poly a = x(1)*y(2); poly b = y(1);
lieBracket(a,b);
↦ -y(1)*x(2)*y(3)+x(1)*y(2)*y(3)
lieBracket(x(1),y(1),2);
↦ -y(1)*x(2)*x(3)*x(4)+3*x(1)*y(2)*x(3)*x(4)-3*x(1)*x(2)*y(3)*x(4)+x(1)*x(2\
   )*x(3)*y(4)
```

7.7.5.12 serreRelations

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

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 "freegb.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 = makeLetterplaceRing(uptodeg);
setring R;
```

```
ideal I = serreRelations(A,1); I = simplify(I,1+2+8);
I;
↦ I[1]=f1(1)*f2(2)*f2(3)-2*f2(1)*f1(2)*f2(3)+f2(1)*f2(2)*f1(3)
↦ I[2]=f1(1)*f3(2)-f3(1)*f1(2)
↦ I[3]=f1(1)*f1(2)*f2(3)-2*f1(1)*f2(2)*f1(3)+f2(1)*f1(2)*f1(3)
↦ I[4]=f2(1)*f3(2)*f3(3)*f3(4)*f3(5)-4*f3(1)*f2(2)*f3(3)*f3(4)*f3(5)+6*f3(1\
  )*f3(2)*f2(3)*f3(4)*f3(5)-4*f3(1)*f3(2)*f3(3)*f2(4)*f3(5)+f3(1)*f3(2)*f3(\
  3)*f3(4)*f2(5)
↦ I[5]=f2(1)*f2(2)*f3(3)-2*f2(1)*f3(2)*f2(3)+f3(1)*f2(2)*f2(3)
```

7.7.5.13 fullSerreRelations

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

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 "freegb.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]=f1(1)*f2(2)*f2(3)-2*f2(1)*f1(2)*f2(3)+f2(1)*f2(2)*f1(3)
↦ fsRel[2]=f1(1)*f1(2)*f2(3)-2*f1(1)*f2(2)*f1(3)+f2(1)*f1(2)*f1(3)
↦ fsRel[3]=e1(1)*e2(2)*e2(3)-2*e2(1)*e1(2)*e2(3)+e2(1)*e2(2)*e1(3)
↦ fsRel[4]=e1(1)*e1(2)*e2(3)-2*e1(1)*e2(2)*e1(3)+e2(1)*e1(2)*e1(3)
↦ fsRel[5]=f2(1)*e1(2)-e1(1)*f2(2)
↦ fsRel[6]=f1(1)*e2(2)-e2(1)*f1(2)
↦ fsRel[7]=-f1(1)*e1(2)+e1(1)*f1(2)-h1(1)
↦ fsRel[8]=-f2(1)*e2(2)+e2(1)*f2(2)-h2(1)
↦ fsRel[9]=h1(1)*h2(2)-h2(1)*h1(2)
↦ fsRel[10]=h1(1)*e1(2)-e1(1)*h1(2)-2*e1(1)
↦ fsRel[11]=f1(1)*h1(2)-h1(1)*f1(2)-2*f1(1)
↦ fsRel[12]=h1(1)*e2(2)-e2(1)*h1(2)+e2(1)
↦ fsRel[13]=f2(1)*h1(2)-h1(1)*f2(2)+f2(1)
↦ fsRel[14]=h2(1)*e1(2)-e1(1)*h2(2)+e1(1)
↦ fsRel[15]=f1(1)*h2(2)-h2(1)*f1(2)+f1(1)
↦ fsRel[16]=h2(1)*e2(2)-e2(1)*h2(2)-2*e2(1)
↦ fsRel[17]=f2(1)*h2(2)-h2(1)*f2(2)-2*f2(1)
```

7.7.5.14 isVar

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: isVar(p); poly p

Return: int

Purpose: check, whether leading monomial of p is a power of a single variable from the basering. Returns the exponent or 0 if p is multivariate.

Example:

```
LIB "freegb.lib";
ring r = 0,(x,y),dp;
poly f = xy+1;
isVar(f);
↦ 0
poly g = y^3;
isVar(g);
↦ 3
poly h = 7*x^3;
isVar(h);
↦ 3
poly i = 1;
isVar(i);
↦ 0
```

7.7.5.15 ademRelations

Procedure from library `freegb.lib` (see Section 7.7.5 [freegb_lib], page 375).

Usage: ademRelations(i,j); i,j int

Return: ring (and exports ideal)

Assume: there are at least i+j variables in the basering

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 "freegb.lib";
def A = ademRelations(2,5);
setring A;
AdemRel;
↦ 6*s(7)*s(0)+s(6)*s(1)
```

7.7.6 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

Theory: Involution is an anti-isomorphism 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

Note: This library provides algebraic tools for computations and operations with algebraic involutions and linear automorphisms of non-commutative algebras

Procedures:

7.7.6.1 findInvo

Procedure from library `involut.lib` (see Section 7.7.6 [involut_lib], page 383).

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

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 7.7.6.2 [findInvoDiag], page 385; Section 7.7.6.5 [involution], page 388.

7.7.6.2 findInvoDiag

Procedure from library `involut.lib` (see Section 7.7.6 [involut_lib], page 383).

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

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 7.7.6.1 [findInvo], page 384; Section 7.7.6.5 [involution], page 388.

7.7.6.3 findAuto

Procedure from library `involut.lib` (see Section 7.7.6 [involut_lib], page 383).

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)

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 7.7.6.1 [findInvo], page 384.

7.7.6.4 ncdetection

Procedure from library `involut.lib` (see Section 7.7.6 [involut_lib], page 383).

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
```

7.7.6.5 involution

Procedure from library `involut.lib` (see Section 7.7.6 [involut_lib], page 383).

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
```

7.7.7 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).

Procedures:

7.7.7.1 GKdim

Procedure from library `gkdim.lib` (see Section 7.7.7 [gkdim_lib], page 389).

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;
↦ //   characteristic : 0
↦ //   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
```

7.7.8 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:

7.7.8.1 makeUsl2

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.17 [makeUg2], page 401; Section 7.7.8.3 [makeUgl], page 392; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.2 makeUsl

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.24 [makeQsl3], page 405; Section 7.7.8.22 [makeQso3], page 404; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.3 [makeUgl], page 392; Section 7.7.8.1 [makeUsl2], page 391.

7.7.8.3 makeUgl

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.4 makeUso5

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.5 makeUso6

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.6 makeUso7

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.7 makeUso8

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.8 makeUso9

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.9 makeUso10

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.10 makeUso11

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
```

```
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.11 makeUso12

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.12 makeUsp1

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393.

7.7.8.13 makeUsp2

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.14 makeUsp3

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2],

page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.15 makeUsp4

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.16 makeUsp5

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.17 makeUg2

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.3 [makeUgl], page 392; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.18 makeUf4

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
```

```
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.19 makeUe6

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.20 makeUe7

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.21 makeUe8

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   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 7.7.8.19 [makeUe6], page 402; Section 7.7.8.20 [makeUe7], page 402; Section 7.7.8.21 [makeUe8], page 403; Section 7.7.8.18 [makeUf4], page 401; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.2 [makeUsl], page 391; Section 7.7.8.4 [makeUso5], page 393; Section 7.7.8.12 [makeUsp1], page 397.

7.7.8.22 makeQso3

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : Q
↦ //   minpoly        : (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 7.7.8.25 [Qso3Casimir], page 407; Section 7.7.8.23 [makeQsl2], page 405; Section 7.7.8.24 [makeQsl3], page 405; Section 7.7.8.17 [makeUg2], page 401; Section 7.7.8.3 [makeUgl], page 392; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.23 makeQsl2

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : (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 7.7.8.24 [makeQsl3], page 405; Section 7.7.8.22 [makeQso3], page 404; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.24 makeQsl3

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : (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 7.7.8.23 [makeQsl2], page 405; Section 7.7.8.22 [makeQso3], page 404; Section 7.7.8.2 [makeUsl], page 391.

7.7.8.25 Qso3Casimir

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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 7.7.8.22 [makeQso3], page 404.

7.7.8.26 GKZsystem

Procedure from library `ncalg.lib` (see Section 7.7.8 [ncalg_lib], page 390).

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 @sc{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;
↦ //   characteristic : 0
↦ //   2 parameter    : b(1) b(2)
↦ //   minpoly        : 0
↦ //   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
print(GKZid);
↦ ‘GKZid‘
// 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);
↦ ‘GKZid‘
```

See also: Section D.4.28 [toric_lib], page 880.

7.7.9 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:

7.7.9.1 CentralQuot

Procedure from library `ncdecomp.lib` (see Section 7.7.9 [ncdecomp_lib], page 409).

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 7.7.9.3 [CenCharDec], page 410; Section 7.7.9.2 [CentralSaturation], page 409.

7.7.9.2 CentralSaturation

Procedure from library `ncdecomp.lib` (see Section 7.7.9 [ncdecomp_lib], page 409).

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 7.7.9.3 [CenCharDec], page 410; Section 7.7.9.1 [CentralQuot], page 409.

7.7.9.3 CenCharDec

Procedure from library `ncdecomp.lib` (see Section 7.7.9 [ncdecomp_lib], page 409).

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 7.7.9.1 [CentralQuot], page 409; Section 7.7.9.2 [CentralSaturation], page 409.

7.7.9.4 IntersectWithSub

Procedure from library `ncdecomp.lib` (see Section 7.7.9 [ncdecomp_lib], page 409).

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)
```

7.7.10 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}

Support: DFG (Deutsche Forschungsgesellschaft) and Metodos algebraicos y efectivos en grupos cuanticos, BFM2001-3141, MCYT, Jose Gomez-Torrecillas (Main researcher).

Main procedures: Auxiliary procedures:

7.7.10.1 Gweights

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
```

```
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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 7.7.10.2 [weightedRing], page 413.

7.7.10.2 weightedRing

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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 7.7.10.1 [Gweights], page 412.

7.7.10.3 ndcond

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   2 parameter    : q1 q2
↦ //   minpoly        : 0
↦ //   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
```

7.7.10.4 Weyl

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 7.7.10.5 [makeWeyl], page 416.

7.7.10.5 makeWeyl

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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 7.7.10.4 [Weyl], page 415.

7.7.10.6 makeHeisenberg

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 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 7.7.10.5 [makeWeyl], page 416.

7.7.10.7 Exterior

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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
```

7.7.10.8 findimAlgebra

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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(j)*x(i) = 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;
↦ //   characteristic : 0
↦ //   2 parameter    : a b
↦ //   minpoly        : 0
↦ //   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]=0
```

7.7.10.9 superCommutative

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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 ].
```

7.7.10.10 rightStd

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

Usage: rightStd(I); I an ideal/ module

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
```

7.7.10.11 moduloSlim

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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
```

7.7.10.12 ncRelations

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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 310; Section 5.1.121 [ringlist], page 211.

7.7.10.13 isCentral

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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)
↦ //   characteristic : 0
↦ //   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
```

7.7.10.14 isNC

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
```

7.7.10.15 isCommutative

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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
```

7.7.10.16 isWeyl

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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
```

7.7.10.17 UpOneMatrix

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
↦ //   noncommutative relations:
↦ //    zx=(q)*xz
```

7.7.10.18 AltVarStart

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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
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;
↦ //   characteristic : 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
```

7.7.10.19 AltVarEnd

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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
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;
↦ //   characteristic : 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
```

7.7.10.20 IsSCA

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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!
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
```

7.7.10.21 makeModElimRing

Procedure from library `nctools.lib` (see Section 7.7.10 [nctools_lib], page 412).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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
```

7.7.11 perron_lib

Library: perron.lib

Purpose: computation of algebraic dependences

Author: Oleksandr Motsak U@D, where U={motsak}, D={mathematik.uni-kl.de}

Procedures:

7.7.11.1 perron

Procedure from library `perron.lib` (see Section 7.7.11 [perron_lib], page 429).

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;
↦ //   characteristic : 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
```

7.7.12 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).

Main procedures: Auxiliary procedures:

7.7.12.1 quantMat

Procedure from library `qmatrix.lib` (see Section 7.7.12 [qmatrix_lib], page 430).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : (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 7.7.12.2 [qminor], page 432.

7.7.12.2 qminor

Procedure from library `qmatrix.lib` (see Section 7.7.12 [qmatrix_lib], page 430).

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 7.7.12.1 [quantMat], page 431.

7.7.12.3 SymGroup

Procedure from library `qmatrix.lib` (see Section 7.7.12 [qmatrix_lib], page 430).

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 7.7.12.5 [LengthSym], page 433; Section 7.7.12.4 [LengthSymElement], page 433.

7.7.12.4 LengthSymElement

Procedure from library `qmatrix.lib` (see Section 7.7.12 [qmatrix_lib], page 430).

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 7.7.12.5 [LengthSym], page 433; Section 7.7.12.3 [SymGroup], page 432.

7.7.12.5 LengthSym

Procedure from library `qmatrix.lib` (see Section 7.7.12 [qmatrix_lib], page 430).

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 7.7.12.4 [LengthSymElement], page 433; Section 7.7.12.3 [SymGroup], page 432.

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.4.18 [primdec_lib], page 779)
- 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 5.1.50 [imap], page 161) - 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;
↦ //   characteristic : 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 49, and Section 3.8.2 [Format of a library], page 54.

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 3.10 [Debugging tools], page 66).

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 470. (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]:
```

```
↦     -2.791287847477920003294023596864
↦ [2]:
↦     1.791287847477920003294023596864
```

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 444.

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 502 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 implictly 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 5.1.21 [division], page 141 for division with remainder in the localization and Section D.5.11.2 [invunit], page 955 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]=12103947791971846719838321886393392913750065060875xy8-28639152114168\
   3198701331939250003266767738632875y38-319544022069090269276462287757356\
   5\
   78554430672591y37+57436621420822663849721381265738895282846320y36+1657764\
   2149487994975739182100310673539324394 00y35+213018481589308191195677223898\
   \
   98682697001205500y34+18221941586630665655859919769615657196480698061 48y33\
   \
   -47017092798928161351569723131963940052201 75y32-13518722696881922676007 86\
   \
   97600850686824231975y31-38730633059298108169615169760250380530011413 75y30\
   \
   +13258866758438740479903820054211440618612900800 00y29+1597720195476063141\
   \
   94679458955424060895269668873 10y28-26270181336309092660633348002625330426\
   \
   7126525y27-75860826908933352690271362489448595447279531 25y26-867853074106\
   \
   49464602285843351672148965395945625y25-55458081432735941021732523311518 35\
   \
   700278863924745y24+19075563013460437364679153779038394895638325y23+548562\
   \
   32271550176105834899677692256107402112 5y22+15746545267764838607395746471 5\
   \
   6810078093398312 5y21-14142791297211762229786542358173595055551911562 50y20\
   \
   -20711190069445893615213399650035715378169943423125y19+272942733337472665\
   \
   57341809297790532298400975 0y18+78906511584533450580184729467741336572095 5\
   \
   3750y17+63554897038491686787729656061044724651089803125y16-22099251729923\
   \
   90669973224476102826607435025596162 5y14+14793713967965590435357948972258 5\
   \
   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]))/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^31-3.200e-04*y^30+1.095\
   e-01*y^29+1.320e+00*y^28-2.170e-05*y^27-6.267e-04*y^26-7.170e-03*y^25-4.5\
```

```
   82e-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 439 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;
↦ //   characteristic : 0
↦ //   1 parameter    : t
↦ //   minpoly        : 0
↦ //   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 `poly.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 MPtcp links

In this example, we demonstrate how MPtcp 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 5.1.135 [stdhilb], page 225) and `stdfglm` (see Section 5.1.134 [stdfglm], page 224) 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 MPtcp 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 = "MPtcp:fork","MPtcp:fork";         // 1.

open(l_fglm); open(l_hilb);

write(l_hilb, quote(system("pid")));                    // 2.
write(l_fglm, quote(system("pid")));
```

```
int pid_hilb,pid_fglm = read(l_hilb),read(l_fglm);

write(l_hilb, quote(stdhilb(i)));                          // 3.
write(l_fglm, quote(stdfglm(eval(i))));

while ((! status(l_hilb, "read", "ready", 1)) &&           // 4.
       (! status(l_fglm, "read", "ready", 1))) {}

if (status(l_hilb, "read", "ready"))
{
  "stdhilb won !!!!"; size(read(l_hilb));
  close(l_hilb); pid_fglm = system("sh","kill "+string(pid_fglm));
}
else                                                       // 5.
{
  "stdfglm won !!!!"; size(read(l_fglm));
  close(l_fglm); pid_hilb = system("sh","kill "+string(pid_hilb));
}
↦ stdfglm won !!!!
↦ 9
```

Some explainatory remarks are in order:

1. Instead of using links of the type `MPtcp:fork`, we alternatively could use `MPtcp:launch` 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. Unfortunately, MPtcp links do not offer means to (asynchronously) interrupt or kill an attached (i.e., launched or forked) process. Therefore, we explicitely need to get the process id numbers of the competing SINGULAR processes, so that we can "kill" the looser later.
3. 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 `MPtcp:launch` 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).
4. We go into a "sleepy" loop and wait until one of the two children finished the computation. That is, the current process checks approximately once per second the status of one of the connecting links, and sleeps (i.e., suspends its execution) in the intermediate time.
5. 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, since it can not be terminated through the link while it is still computing.

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. Detailed information on the latter topic can be found in a separate PostScript file at `http://www.singular.uni-kl.de/DynMod.ps`.

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("kstd.so");
listvar(package);
↦ // Kstd                 [0]  package (C,kstd.so)
↦ // Standard             [0]  package (S,standard.lib)
↦ // Top                  [0]  package (N)

// set the number of components to be considered to 1
module mostd=kstd(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=Kstd::kstd(mo,2); // calling procedure in Kstd
                    // 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

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 `st`andard basis). The computation of reduced Groebner and standard bases may be forced by setting `option(redSB)` (see Section 5.1.98 [option], page 192).

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 447, Section A.2.3 [slim Groebner bases], page 449. 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 442). We set `option(prot)` to make SINGULAR display some information on the performed computations (see Section 5.1.98 [option], page 192 for an interpretation of the displayed symbols). For long running computations, it is always recommended to set this option.

```
LIB "poly.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);
↦ [31:1]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(7\
   3)--------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(21)---\
   ------27-s(11)28-s(13)--29-s(12)-30--s--31-s(11)---32-s33(7)s(10)---34-s-\
   35----[1023:2]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)
↦ 31
size(sI);
↦ 17
t=timer;
sI=groebner(I);
↦ compute hilbert series with std in ring (32003),(a,b,c,d,e,f,@),(dp(7),C)
↦ weights used for hilbert series: 1,1,1,1,1,1,1
↦ [15:1]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(7\
   1)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(116)s\
   (121)s(126)s(128)s(132)-------------------------------(100)------------\
```

```
   -11-s(87)---------------------------------------12-s(50)--------13-s(44)s\
   (47)s(51)s(55)-------------14-s(45)s(48)s(51)s(55)s(58)s(61)s(64)s(67)s(7\
   0)--------------------15-s(52)s(55)s(58)s(61)s(64)s(67)s(70)s(73)s(76)s(7\
   9)s(82)-----------------------------------16--------------------------\
   -----------------17-
↦ product criterion:284 chain criterion:4184
↦ std with hilb in (32003),(a,b,c,d,e,f,@),(lp(6),dp(1),C)
↦ [15:1]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)----------shhhhhhhhhhhhhhhhhhhhhh\
   8(134)s(136)s(139)s(142)s(145)s(147)s(150)s(151)s(153)s(155)s(158)s(161)s\
   (163)s(166)s(169)s(172)s(175)s(177)s(180)s(183)----------------------s(\
   171)s(178)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh9(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)---------shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh\
   hhhhhhhh10(163)s(166)s(168)s(171)s(177)s(180)s(183)s(186)shhhhhhhhhhhhhh\
   hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh11(125)s(128)s(130)s\
   (133)s(136)shhhhhhhhhhhhhhhhhhhhhhhhhhhh12(110)s(113)s(120)s(123)s(127)-\
   ------------shhhhhhhhhhhhhhh13(102)s(106)s(109)s(111)s(114)s(117)----shhh\
   hhhhhhhhhhhhhhhhhhhhhhhhhhhh14(85)s(90)s(93)s(97)s(100)s(103)---(100)-s(1\
   03)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh15(68)s(72)s(75)s(79)s(85)----\
   shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh16(42)s(45)s(49)shhhhhhhhhhh\
   hhhhhhhhh17(34)s(37)shhhhhhhhhhhhh18(27)s(30)s(32)[255:2]-shhhhhhhh19(\
   26)s(29)s(32)shhhhhhhhhhhh20(22)s(25)s(28)shhhhhhhhhhhh21(20)s(26)shhhh\
   hhhhhhh22(18)shhhhhhhhh23(12)shhhhh24(11)s(14)-shhhh25(13)s(18)-s(21)shh\
   hhhh26(18)shhhhhhhhhhh27(9)shhhhh28(8)shhhh29(7)shhhh30(8)-shhh31shhhhh\
   32(7)shhhh33shhhh34(6)shhhhhhhh36(2)s37(6)shhhh38shhhh39shhhhhhh42(2)s43\
   (5)shhhh44shhhhhhh48s49shhhh50shhhhhhh54shhhhh
↦ product criterion:720 chain criterion:11620
↦ hilbert series criterion:532
↦ dehomogenization
↦ simplification
↦ imap to ring (32003),(a,b,c,d,e,f),(lp(6),C)
timer-t;                                  // used time (in 1/100 secs)
↦ 12
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 5.1.134 [stdfglm], page 224) and variants of the Groebner walk conversion algorithm (which works for arbitrary ideals, See Section 5.1.40 [frwalk], page 153, Section D.4.6 [grwalk_lib], page 681). An implementation of the Hilbert-driven Buchberger algorithm is accessible via the `stdhilb` command (see also Section 5.1.133 [std], page 223).

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 (32003),(a,b,c,d,e),(dp(5),C)
↦ [63:2]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(15)s(1\
  7)--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 (32003),(a,b,c,d,e,@),(dp(6),C)
↦ weights used for hilbert series: 1,1,1,1,1,1
↦ [31:1]1(4)s2(3)s3(2)s4ss5(3)s(4)s(5)-s6s(6)s(7)s(9)s(11)-7-ss(13)s(15)s(1\
  7)--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 (32003),(a,b,c,d,e,@),(lp(5),dp(1),C)
↦ [31:1]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)--------shhhhh\
  hhhhhhhhhhhhhh9(51)s(53)s(56)s(58)s(61)s(64)------s(61)s(64)shhhhhhhhhhhh\
  hhh10(53)s(55)s(58)s(62)s(64)s(67)s(70)--s(71)------s(68)s(71)s(73)--shhh\
  hhhhhhhhhhhhh11(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)shhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh14(3\
  1)s(33)s(36)s(39)s(42)s(45)shhhhhhhhhhhhhhhhhhhhhhh15(23)s(26)s(29)s(32\
```

```
   )s(35)shhhhhhhhhhhhhhhhhh16(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)shhhhh\
   h26(7)s(10)shhhhhh27(7)s(10)shhhhhh28(7)s(10)shhhhhh29(7)s(10)shhhhhh30(7\
   )s(9)shhhhhh31(6)shhhhhh[1023:2]32(3)shhh33shhh34shhh35shhh36shhh37shhh38\
   shhh39shhh40shhh41shhh42shhh43shhh44shhh45shhh46shhh47shhh48shhh49shhh50s\
   hhh51shhh52shhh53shhh54shhhhhh
↦ product criterion:491 chain criterion:11799
↦ hilbert series criterion:417
↦ dehomogenization
↦ simplification
↦ imap to ring (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);
↦ [63:1]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(10)s(1\
   1)s(13)s8(12)s(13)s(15)s.s(14).s.9.s(16)s(17)s(19)........10.s(20).s(21)s\
   s..11.s(23)s(25).ss(27)...s(28)s(26)...12.s(25)sss(23)sss.......s(22)...1\
   3.s(23)ssssssss(21)s(22)sssss(21)ss..14.ss(22)s.s.sssss(21)s(22)sss.s...1\
   5.ssss(21)s(22)sssssssssss(21)s(22)sss16.ssssssss(21)s(22)sssssssssss17ss(\
   21)s(22)sssssssssss(21)sss(22)ss(21)ss18(22)s(21)s(22)s.s.............19.\
   sssss(21)ss(22)sssssssssss(21)s(22)s20.sssssssssss(21)s........21.s(22)ssss\
   sssssssssss(21)s(22)ssss22sssssssssssss(21)s(22)sssssss23sssssssssss(21)s(\
   22)ssssssss24sssssssssssss(21)s(22)sssssss25sssssssssss(21)s(22)sssssssss26\
   sssssssssss(21)s(20)ssssssss27.sssssssss..........s28.ssssss.............2\
   9.sssssssssssssssssss30sssssssssssssssssss31.sssssssssssssssssss32.ssssss\
   sssssssssssss33sssssssssssssssssss34sssssssssssssssssss35sssssssssssss\
   sssssss36sssssssssssssssssss37sssssssssssssssssss38sssssssssssssssssss\
   39sssssssssssssssssss40sssssssssssssssssss41ssss---------------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 developement was developed in the diploma thesis of Michael Brickenstein. It has been published as `http://www.mathematik.uni-kl.de/~zca/Reports_on_ca/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 5.1.127 [slimgb], page 218.

A.3 Commutative Algebra

A.3.1 Saturation

For any two ideals i, j in the basering R let

$$\mathrm{sat}(i,j) = \{x \in R \mid \exists\, n \text{ s.t. } x \cdot (j^n) \subseteq i\} = \bigcup_{n=1}^{\infty} i : j^n$$

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 471). 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]=1
↦ [2]:
↦    17
  // 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;
↦ //   # ground field : 9
↦ //   primitive element : 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 506), 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 447):

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]=x
↦ j2[2]=z2-y3
  // 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+12ac2e-2abde
  // 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
↦ // ** too many arguments for nselect
  j;                             // containing variable 1,...,4
↦ j[1]=3c2-4bd+ae
↦ j[2]=2bcd-3ad2-3b2e+4ace
↦ j[3]=8b2d2-9acd2-9b2ce+12ac2e-2abde
↦ j[4]=2xd2e-2xce2-8yd3+11ycde-3ybe2
↦ j[5]=2xcde-2xbe2-8ycd2+3yc2e+8ybde-3yae2
↦ j[6]=2xbde-2xae2-8ybd2+9ybce-yade
↦ j[7]=xbce-xade-4ybcd+4yad2+6yb2e-6yace
↦ j[8]=2xb2e-2xace+6ybc2-8yb2d-yacd+3yabe
↦ j[9]=16xb2d-18xacd+2xabe+15yac2-24yabd+9ya2e
↦ j[10]=4xb2c-6xabd+2xa2e-yabc+ya2d
↦ j[11]=8xb3-11xabc+3xa2d-2yab2+2ya2c
↦ j[12]=x2e2-2xyde-8y2d2+9y2ce
↦ j[13]=x2de-4xyd2+2xyce-2y2cd+3y2be
↦ j[14]=x2ce-4xycd+2xybe+6y2c2-8y2bd+3y2ae
↦ j[15]=x2be-4xybd+2xyae+y2ad
↦ j[16]=x2ae-4xybc+2xyad+y2ac
↦ j[17]=2x2bc-3x2ad+4xyb2-2xyac-y2ab
↦ j[18]=8x2b2-9x2ac+2xyab-y2a2
↦ j[19]=4t3y-e
↦ j[20]=x4e-4x3yd+6x2y2c-4xy3b+y4a
```

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 5.1.118 [res], page 209
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.

`lres` Section 5.1.74 [lres], page 177
computes a free resolution of an ideal or module with LaScala's method. The input needs to be homogeneous.

`mres` Section 5.1.87 [mres], page 186
computes a minimal free resolution of an ideal or module with the syzygy method.

`sres` Section 5.1.131 [sres], page 221
computes a free resolution of an ideal or module with Schreyer's method. The input has to be a standard basis.

`nres` Section 5.1.94 [nres], page 190
computes a free resolution of an ideal or module with the standard basis method.

`minres` Section 5.1.82 [minres], page 184
minimizes a free resolution of an ideal or module.

`syz` Section 5.1.138 [syz], page 229
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;          // convertion 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
↦ resolution not minimized yet
↦
  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 460).

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 Extˆ1(R/m,R/m)
↦ // dimension of Extˆ1:  0
↦ // vdim of Extˆ1:       2
↦
  print(T1K);
↦ 0,x,0,y,
↦ x,0,y,0
  printlevel=2;             // gives more explanation
  module T2K=Ext(2,m,m);    // computes Extˆ2(R/m,R/m)
↦ // Computing Extˆ2 (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 Extˆ2:  0
↦ // vdim of Extˆ2:       2
↦
  print(std(T2K));
↦ 0,x,0,y,
↦ x,0,y,0
  printlevel=0;
  module E = Ext(1,syz(m),syz(m));
↦ // dimension of Extˆ1:  0
↦ // vdim of Extˆ1:       2
↦
  print(std(E));
↦ x, 0,-y2,x,0,y,
↦ -y,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)/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]=x8+y4
↦    _[3]=x2+y2
↦ [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]=x2+y
↦    _[3]=x+y
↦ [2]:
↦    1,4,2
  // 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]=x4+(-i)*y2
↦    _[3]=x4+(i)*y2
↦    _[4]=x+(-i)*y
↦    _[5]=x+(i)*y
↦ [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

$$\text{milnor}(f) = \dim_K(K[[x_1,\ldots,x_n]]/\text{jacob}(f)),$$

respectively

$$\text{tjurina}(f) = \dim_K(K[[x_1,\ldots,x_n]]/((f)+\text{jacob}(f)))$$

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 $\text{Loc}_{(x)}K[x_1,\ldots,x_n]$, 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 502 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.5.13 [sing_lib], page 961 for the library commands for the computation of the Milnor and Tjurina number.

```
option(prot);
ring r1 = 32003,(x,y,z),ds;
r1;
↦ //   characteristic : 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);
↦ [1023:2]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);
↦ [1023:2]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)(9)sH(20)16(10).17...........18.......19..----.\
   .sH(19)
↦ product criterion:10 chain criterion:53
  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);
↦ [1023:2]7(2)s8s10s11s12s13(3)ss(4)s14(5)s(6)s15(7).....s(8)16.s...s(9)..1\
   7...........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
↦ [1023:2]7(16)s8(15)s10s11ss(16)-12.s-s13s(17)s(18)s(19)-s(18).-14-s(17)-s\
   (16)ss(17)s15(18)..-s...--.16....-.......s(16).sH(23)s(18)...17..........\
   18.........sH(20)17(17).....................18..........19..---....-.-....\
   .....20.-----...s17(9).........18.............19..-.......20.-......21..\
   .......sH(19)16(5).....18......19.-----
↦ 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 470 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 470) 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)[1];           // 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)[1];             // 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
↦ //         4 t^0
↦ //       -20 t^1
↦ //        40 t^2
↦ //       -40 t^3
↦ //        20 t^4
↦ //        -4 t^5
↦
↦ //         4 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 474 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: (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]=xy+Ez+F
↦ Fs[1,2]=y2+z2+x3+Ay+Bx2+Cx+D
  // 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.
↦ // ** J is no standard basis
↦
↦
↦ // '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.5.9 [hnoether_lib], page 935) 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]=x-z
↦       _[2]=z2-y
↦    [2]:
↦       _[1]=x-z
↦       _[2]=z2-y
↦ [2]:
↦    [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
  def li=_[2];
  ideal i2=li[2];       // call the second ideal i2
  // 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]=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
↦ [2]:
↦    [1]:
↦       _[1]=-z2+y
↦       _[2]=x-z
↦    [2]:
↦       _[1]=-z2+y
↦       _[2]=x-z
  // 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 first branch
  // is smooth, only the second ideal can be the one which can be decomposed
  // further.
  // Let us now consider the normalization of this second ideal i2.
  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]:
↦       //   characteristic : 0
↦ //   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 `http://www.singular.uni-kl.de/tutor_resol.ps`.

```
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 Non-commutative Algebra

A.6.1 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 306). The command `twostd` 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], page 309).

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 310).
Further on, we use the command `reduce` (see Section 7.3.23 [reduce (plural)], page 301) 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 5.1.98 [option], page 192 and Section 7.4.2 [Groebner bases in G-algebras], page 312 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 5.1.98 [option], page 192 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.6.2 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;
↦ //   characteristic : 0
↦ //   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
↦ //   characteristic : 0
↦ //   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.7 Applications

A.7.1 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 445, resp. Section D.4.18.7 [radical], page 783).

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 5.1.55 [interred], page 165). 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.7.3 [triang_lib], page 1046) for the ideal I, or by applying the factorizing Buchberger algorithm (see Section 5.1.29 [facstd], page 146), or by computing a primary decomposition of I (see Section D.4.18 [primdec_lib], page 779).

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.4.5.7 [sat], page 679).

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)[1];  // 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*2.44948974278317809819728407470 6)
↦    [5]:
↦       (i*1.73205080756887729352744634150 6)
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
↦    [4]:
↦       (-i*2.44948974278317809819728407470 6)
↦    [5]:
↦       (i*1.73205080756887729352744634150 6)
```

```
↦ [3]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
↦    [4]:
↦       (i*2.44948974278317809819728407 4706)
↦    [5]:
↦       (-i*1.73205080756887729352744634 1506)
↦ [4]:
↦    [1]:
↦       1
↦    [2]:
↦       1
↦    [3]:
↦       1
↦    [4]:
↦       (-i*2.44948974278317809819728407 4706)
↦    [5]:
↦       (-i*1.73205080756887729352744634 1506)
↦ 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*2.44948974278317809819728407 4706)
↦ [5]:
↦    (-i*1.73205080756887729352744634 1506)
```

A.7.2 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 **
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 29). 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 503). 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 5.1.1 [attrib], page 127, Section 5.1.95 [nrows], page 191) 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 $N = R^n/\text{std}(M)$). 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 127 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 5.1.118 [res], page 209; Section C.3 [Syzygies and resolutions], page 508).

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 5.1.115 [reduce], page 206).

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 inhomogenous 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 523.

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 29). 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: reverse 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:

$x^a < x^b \Leftrightarrow \exists\ 1 \leq i \leq n : M_1 a = M_1 b, \ldots, M_{i-1} a = M_{i-1} b$ 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):

lp: $\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$ dp: $\begin{pmatrix}1&1&1\\0&0&-1\\0&-1&0\end{pmatrix}$ Dp: $\begin{pmatrix}1&1&1\\1&0&0\\0&1&0\end{pmatrix}$

wp(1,2,3): $\begin{pmatrix}1&2&3\\0&0&-1\\0&-1&0\end{pmatrix}$ Wp(1,2,3): $\begin{pmatrix}1&2&3\\1&0&0\\0&1&0\end{pmatrix}$

ls: $\begin{pmatrix}-1&0&0\\0&-1&0\\0&0&-1\end{pmatrix}$ ds: $\begin{pmatrix}-1&-1&-1\\0&0&-1\\0&-1&0\end{pmatrix}$ Ds: $\begin{pmatrix}-1&-1&-1\\1&0&0\\0&1&0\end{pmatrix}$

ws(1,2,3): $\begin{pmatrix}-1&-2&-3\\0&0&-1\\0&-1&0\end{pmatrix}$ 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:

(dp(3), wp(1,2,3)): $\begin{pmatrix}1&1&1&0&0&0\\0&0&-1&0&0&0\\0&-1&0&0&0&0\\0&0&0&1&2&3\\0&0&0&0&0&-1\\0&0&0&0&-1&0\end{pmatrix}$

(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:

(dp(3), a(1,2,3),dp(3)): $\begin{pmatrix}1&1&1&0&0&0\\0&0&-1&0&0&0\\0&-1&0&0&0&0\\0&0&0&1&2&3\\0&0&0&1&1&1\\0&0&0&0&0&-1\\0&0&0&0&-1&0\end{pmatrix}$

(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;
↦ //   characteristic : 0
↦ //   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);
```

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 523).

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 $L(I)$.

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 $g \in G$. (Note that such a function is not unique).

  $\mathrm{NF}(p|G)$ 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 $\mathrm{M} = \bigoplus_{i \in Z} M_i$ 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, $\mathrm{syz}(I)$, 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

$$R^r = \bigoplus_a R \cdot e_{a,0} \xleftarrow{A_1} \bigoplus_a R \cdot e_{a,1} \longleftarrow \ldots \longleftarrow \bigoplus_a R \cdot e_{a,n} \longleftarrow 0$$

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

$$\deg(e_{a,j}) \leq s + j - 1 \quad \text{for all } j.$$

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 $\mathrm{lvar}(f)$ (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 $k \leq n$ then $\mathrm{lvar}(f) = x_k$.

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 $\exp(2\pi iM)$ 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,   0,   1,  -2/5,
↦ 0,    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,
↦ 2xy2+5x4,
↦ 5x5-5y5,
↦ 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]` (seperated 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 $x^3+y^3+z^3+\lambda xyz=0$, $\lambda^3 \neq -27$. 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 D.4.9 [lll_lib], page 710, see see [[Coh93]], page 517). 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.4.28 [toric_lib], page 880.

C.6.2.1 The algorithm of Conti and Traverso

The algorithm of Conti and Traverso (see [[CoTr91]], page 517) 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 516).

C.6.2.2 The algorithm of Pottier

The algorithm of Pottier (see [[Pot94]], page 517) starts by computing a lattice basis $v_1, \ldots, v_r$ for the integer kernel of A using the LLL-algorithm (Section D.4.9 [lll_lib], page 710). 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 517) 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 517), 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 517) 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 517) 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 517) or the use of bit vectors to represent the support of a monomial (see [[Big97]], page 517) 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 517 and see [[The99]], page 517.

An implementation of the above algorithm and some examples can be found in Section D.4.8 [intprog_lib], page 708.

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 517.

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 310.

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.9.2 [decodegb_lib], page 1090. More details can be obtained from [[BP2008b]], page 523.

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 $f \in CF : f(syndrome(\mathbf{r}), x) = poly(x)$. The roots of $poly(x) = 0$ 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 $S_C = \{i_1, \ldots, i_{n-k}\}$ 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

$$s_{i_m} = r(a^{i_m}) = e(a^{i_m}) = \sum_{l=1}^{t} e_{j_l} (a^{i_m})^{j_l}, \quad 1 \leq m \leq n - k,$$

where t is the number of errors, $j_1, \ldots, j_t$ are the **error positions** and $e_{j_1}, \ldots, e_{j_t}$ are the **error values**. Define $z_l = a^{j_l}$ and $y_l = e_{j_l}$. Then $z_1, \ldots, z_t$ are the error locations and $y_1, \ldots, y_t$ are the error values and the syndromes above become **generalized power sum functions** $s_{i_m} = \sum_{l=1}^{t} y_l z_l^{i_m}$, $1 \leq m \leq n - k$.

CRHT-ideal

Replace the concrete values above by variables and add some natural restrictions. Introduce

- $f_u := \sum_{l=1}^{e} Y_l Z_l^{i_u} - X_u = 0, 1 \leq u \leq n-k$;
- $\epsilon_j := X_j^{q^m} - X_j = 0, 1 \leq j \leq n-k$, since $s_j \in F$;
- $\eta_i := Z_i^{n+1} - Z_i = 0, 1 \leq i \leq e$, since a^{j_i} are either n-th roots of unity or zero;
- $\lambda_i := Y_i^{q-1} - 1 = 0, 1 \leq i \leq e$, since $y_l \in F_q \setminus \{0\}$.

We obtain the following set of polynomials in the variables $X = (X_1, \ldots, X_{n-k})$, $Z = (Z_1, \ldots, Z_e)$ and $Y = (Y_1, \ldots, Y_e)$:

$$F_C = \{f_j, \epsilon_j, \eta_i, \lambda_i : 1 \leq j \leq n-k, 1 \leq i \leq e\} \subset F_q[X, Z, Y].$$

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 $F_q[X_1, \ldots, X_{n-k}, Z]$ that satisfies the following two properties:

- $L_C = Z^e + a_{t-1}Z^{e-1} + \ldots + a_0$ with $a_j \in F_q[X_1, \ldots, X_{n-k}]$, $0 \leq j \leq e-1$, where e is the error-correcting capacity;
- given a syndrome $\mathbf{s} = (s_{i_1}, \ldots, s_{i_{n-k}}) \in F^{n-k}$ corresponding to an error of weight $t \leq e$ and error locations $\{k_1, \ldots, k_t\}$, if we evaluate the $X_u = s_{i_u}$ for all $1 \leq u \leq n-k$, then the roots of $L_C(\mathbf{s}, Z)$ are exactly $a^{k_1}, \ldots, a^{k_t}$ 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.9.2 [decodegb_lib], page 1090. More on Cooper's philosophy and the general error-locator polynomial can be found in [[OS2005]], page 523.

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:

$$\begin{gathered} Z_1^{i_1} + \ldots + Z_w^{i_1} = 0, \\ \vdots \\ Z_1^{i_v} + \ldots + Z_w^{i_v} = 0, \\ Z_1^n - 1 = 0, \\ \vdots \\ Z_w^n - 1 = 0, \end{gathered}$$

$$p(n, Z_i, Z_j) = 0, 1 \leq i < j \leq w.$$

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.9.2 [decodegb_lib], page 1090. More on finding the minimum distance with Groebner bases can be found in [[S2007]], page 523. See [[OS2005]], page 523, 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.9.2 [decodegb_lib], page 1090. More on this method and the method based on Waring function can be found in [[ABF2002]], page 523. See also [[ABF2008]], page 523.

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 $t \leq e$. 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

$$\sum_{j=1}^t E_j f_i(X_{j1}, \ldots, X_{js}) - s_i, 1 \leq i \leq n - k,$$

$$g_l(X_{j1}, \ldots, X_{js}), 1 \leq l \leq m,$$

$$E_k^{q-1} - 1.$$

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.9.2 [decodegb_lib], page 1090. More on this method can be found in [[FL1998]], page 523.

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 $\mathbf{x} * \mathbf{y} = (x_1 y_1, \ldots, x_n y_n)$. 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 $n \leq q$. 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 $\mathbf{b}_j \cdot \mathbf{e} = u_j(\mathbf{e})$.

Contructing 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.9.2 [decodegb_lib], page 1090. More on this method can be found in [[BP2008a]], page 523.

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 `http://www.mathematik.uni-kl.de/~zca/Reports_on_ca`. Other sources to check are `http://symbolicnet.org/`, `http://www-sop.inria.fr/galaad/`, `http://www.bath.ac.uk/~masjpf/CAIN.html`,... and the following list of books.

For references on non-commutative algebras and algorithms, see Section 7.4.4 [References (plural)], page 314.

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 5.1.70 [LIB], page 174) for loading of single libraries, and the command `LIB "all.lib";` for loading all libraries.

See also Section 7.7 [Non-commutative libraries], page 320.

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 525).

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 525).

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;
↦ //   characteristic : 0
↦ //   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);
↦ Id(1)[1]=-x
↦ Id(1)[2]=y+z+u+v-1
↦ Id(1)[3]=yzuvx-1
↦ Id(1)[4]=x2+1
setring R;
module m = x3*[1,1,1], (xyzuv-1)*[1,0,1];
def Q = par2varRing(m)[1]; Q;
↦ //   characteristic : 0
↦ //   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 (12939,2010-07-01)
↦ // ** loaded qmatrix.lib (12231,2009-11-02)
↦ // ** loaded perron.lib (12231,2009-11-02)
↦ // ** loaded nctools.lib (12790,2010-05-14)
↦ // ** loaded ncdecomp.lib (12790,2010-05-14)
↦ // ** loaded ncalg.lib (12231,2009-11-02)
↦ // ** loaded involut.lib (12790,2010-05-14)
↦ // ** loaded gkdim.lib (12235,2009-11-03)
↦ // ** loaded freegb.lib (12790,2010-05-14)
↦ // ** loaded dmodvar.lib (12790,2010-05-14)
↦ // ** loaded dmodapp.lib (12790,2010-05-14)
↦ // ** loaded dmod.lib (12790,2010-05-14)
↦ // ** loaded central.lib (12387,2009-12-14)
↦ // ** loaded bfun.lib (12790,2010-05-14)
↦ // ** loaded zeroset.lib (12853,2010-06-10)
↦ // ** loaded weierstr.lib (12231,2009-11-02)
↦ // ** loaded tropical.lib (12812,2010-05-21)
↦ // ** loaded triang.lib (12231,2009-11-02)
↦ // ** loaded toric.lib (12330,2009-11-26)
↦ // ** loaded teachstd.lib (12231,2009-11-02)
↦ // ** loaded surfex.lib (12231,2009-11-02)
↦ // ** loaded surf.lib (12790,2010-05-14)
↦ // ** loaded stratify.lib (12231,2009-11-02)
↦ // ** loaded spectrum.lib (12231,2009-11-02)
↦ // ** loaded spcurve.lib (12231,2009-11-02)
↦ // ** loaded solve.lib (12231,2009-11-02)
↦ // ** loaded signcond.lib
↦ // ** loaded sing4ti2.lib (12231,2009-11-02)
↦ // ** loaded sing.lib (12394,2010-01-04)
↦ // ** loaded sheafcoh.lib (12933,2010-06-30)
↦ // ** loaded sagbi.lib (12381,2009-12-11)
↦ // ** loaded rootsur.lib
↦ // ** loaded rootsmr.lib
↦ // ** loaded rinvar.lib (12231,2009-11-02)
↦ // ** loaded ring.lib (12231,2009-11-02)
↦ // ** loaded reszeta.lib (12231,2009-11-02)
↦ // ** loaded resolve.lib (12387,2009-12-14)
↦ // ** loaded resgraph.lib (12231,2009-11-02)
↦ // ** loaded resbin.lib $Id: resbin.lib$
↦ // ** loaded reesclos.lib (1.32,2001/01/16)
↦ // ** loaded redcgs.lib (12231,2009-11-02)
↦ // ** loaded realrad.lib (12529,2010-02-08)
↦ // ** loaded random.lib (12827,2010-05-28)
↦ // ** loaded qhmoduli.lib (12790,2010-05-14)
↦ // ** loaded primitiv.lib (12860,2010-06-15)
↦ // ** loaded primdec.lib (12617,2010-03-08)
↦ // ** loaded presolve.lib (12790,2010-05-14)
↦ // ** loaded polymake.lib (12231,2009-11-02)
↦ // ** loaded poly.lib (12443,2010-01-19)
↦ // ** loaded pointid.lib (12231,2009-11-02)
↦ // ** loaded phindex.lib (12571,2010-03-01)
↦ // ** loaded ntsolve.lib (12231,2009-11-02)
↦ // ** loaded normaliz.lib (12790,2010-05-14)
```

```
↦ // ** loaded normal.lib (12801,2010-05-20)
↦ // ** loaded noether.lib (12231,2009-11-02)
↦ // ** loaded mregular.lib (12387,2009-12-14)
↦ // ** loaded mprimdec.lib (12790,2010-05-14)
↦ // ** loaded monomial.lib $Id: monomial.lib$
↦ // ** loaded mondromy.lib (12231,2009-11-02)
↦ // ** loaded modstd.lib (12868,2010-06-16)
↦ // ** loaded matrix.lib (12898,2010-06-23)
↦ // ** loaded makedbm.lib (12231,2009-11-02)
↦ // ** loaded lll.lib (12231,2009-11-02)
↦ // ** loaded linalg.lib (12258,2009-11-09)
↦ // ** loaded latex.lib (12231,2009-11-02)
↦ // ** loaded kskernel.lib (12231,2009-11-02)
↦ // ** loaded jacobson.lib (12790,2010-05-14)
↦ // ** loaded intprog.lib (12231,2009-11-02)
↦ // ** loaded inout.lib (12541,2010-02-09)
↦ // ** loaded hyperel.lib (12231,2009-11-02)
↦ // ** loaded homolog.lib (12381,2009-12-11)
↦ // ** loaded hnoether.lib (12231,2009-11-02)
↦ // ** loaded grwalk.lib (12790,2010-05-14)
↦ // ** loaded groups.lib (12231,2009-11-02)
↦ // ** loaded graphics.lib (12231,2009-11-02)
↦ // ** loaded gmssing.lib (12790,2010-05-14)
↦ // ** loaded gmspoly.lib (12529,2010-02-08)
↦ // ** loaded general.lib (12904,2010-06-24)
↦ // ** loaded finvar.lib (12481,2010-01-29)
↦ // ** loaded equising.lib (12387,2009-12-14)
↦ // ** loaded elim.lib (12231,2009-11-02)
↦ // ** loaded deform.lib (12231,2009-11-02)
↦ // ** loaded decodegb.lib (12790,2010-05-14)
↦ // ** loaded curvepar.lib (12231,2009-11-02)
↦ // ** loaded crypto.lib (12529,2010-02-08)
↦ // ** loaded control.lib (12790,2010-05-14)
↦ // ** loaded compregb.lib (12231,2009-11-02)
↦ // ** loaded classify.lib (12231,2009-11-02)
↦ // ** loaded cimonom.lib (12231,2009-11-02)
↦ // ** loaded brnoeth.lib (12529,2010-02-08)
↦ // ** loaded atkins.lib (12932,2010-06-30)
↦ // ** redefining primeList **
↦ // ** loaded assprime.lib (12938,2010-07-01)
↦ // ** loaded arcpoint.lib (12231,2009-11-02)
↦ // ** loaded algebra.lib (12436,2010-01-15)
↦ // ** loaded alexpoly.lib (12231,2009-11-02)
↦ // ** loaded aksaka.lib (12932,2010-06-30)
↦ // ** loaded ainvar.lib (12754,2010-04-29)
↦ // ** loaded absfact.lib (12231,2009-11-02)
```

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 528).

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 528).

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 537).

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);
A_Z("Z",-5);
string sR = "ring R = (0,"+A_Z("A",6)+"),("+A_Z("a",10)+"),dp;";
sR;
execute(sR);
R;
```

D.2.3.2 ASCII

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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();"";
ASCII(42);
ASCII(32,126);
```

D.2.3.3 absValue

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);
poly f=-4;
absValue(f);
```

D.2.3.4 binomial

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: binomial(n,k); n,k integers

Return: binomial(n,k); binomial coefficient n choose k
- of type bigint (computed in characteristic 0)

Note: In any characteristic, binomial(n,k) = coefficient of x^k in (1+x)^n

Example:

```
LIB "general.lib";
binomial(200,100);"";                  //type bigint
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
```

See also: Section 5.1.105 [prime], page 199.

D.2.3.5 deleteSublist

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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;
```

D.2.3.6 factorial

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: factorial(n); n integer

Return: factorial(n): n! of type bigint.

Example:

```
LIB "general.lib";
factorial(37);"";                  //37! (as long integer)
```

See also: Section 5.1.105 [prime], page 199.

D.2.3.7 fibonacci

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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); "";              //f(42) of type string (as long integer)
ring r = 2,x,dp;
number b = fibonacci(42,2);     //f(42) of type number, computed in r
b;
```

See also: Section 5.1.105 [prime], page 199.

D.2.3.8 kmemory

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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();
kmemory(1,1);
```

D.2.3.9 killall

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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();
killall("ring");            // kills all rings
listvar();
killall("not", "int");      // kills all variables except int's (and procs)
listvar();
killall();                  // kills all vars except loaded procs
listvar();
```

D.2.3.10 number_e

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);"";
ring R = 0,t,lp;
number e = number_e(30);
e;
```

D.2.3.11 number_pi

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);"";
ring r = (real,10),t,dp;
number pi = number_pi(11); pi;
```

D.2.3.12 primes

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);"";
intvec v = primes(37,1); v;
```

D.2.3.13 product

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);
product(m[2..3]);
matrix M[2][3] = 1,x,2,y,3,z;
product(M);
intvec v=2,4,6;
product(M,v);
intvec iv = 1,2,3,4,5,6,7,8,9;
v=1..5,7,9;
product(iv,v);
intmat A[2][3] = 1,1,1,2,2,2;
product(A,3..5);
```

D.2.3.14 sort

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: sort(id[,v,o,n]); id = ideal/module/intvec/list(of intvec's or int's)
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 of intvec's or int's: consider a list element, say id[1]=3,2,5, as exponent vector of the monomial x^3*y^2*z^5; the corresponding monomials are ordered w.r.t. intvec v (s.a.). If no v is present, the monomials are sorted w.r.t. ordering o (if o is given) or w.r.t. lexicographical ordering (if no o is given). The corresponding ordered list of exponent vectors is returned.
(e.g. sort(id); sorts lexicographically, smaller int's come first)
WARNING: Since negative exponents create the 0 polynomial in Singular, id should not contain negative integers: the result might not be as expected
- if id=intvec: id is treated as list of integers
- 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
sort(i,3..1);
sort(i,"ls")[1];     //sort w.r.t. negative lex ordering
intvec v =1,10..5,2..4;v;
sort(v)[1];          // sort v lexicographically
sort(v,"Dp",1)[1];   // sort v w.r.t (total sum, reverse lex)
// Note that in general: lead(sort(M)) != sort(lead(M)), e.g:
module M = [0, 1, 1, 0], [1, 0, 0, 1]; M;
sort(lead(M), "c, dp")[1];
lead(sort(M, "c, dp")[1]);
// 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] ] );
sort( M, sort(lead(M), "c,dp")[2] )[1];
```

D.2.3.15 sum

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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);
sum(pv,2..5);
matrix M[2][3] = 1,x,2,y,3,z;
intvec w=2,4,6;
sum(M,w);
intvec iv = 1,2,3,4,5,6,7,8,9;
sum(iv,2..4);
iv = intvec(1..100);
sum(iv);
ring r2 = 0,(x(1..10)),dp;
sum(x(3..7),intvec(1,3,5));
```

D.2.3.16 watchdog

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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 MP package must be enabled
* 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";
ring r=0,(x,y,z),dp;
poly f=x^30+y^30;
watchdog(1,"factorize(eval("+string(f)+"))");
watchdog(100,"factorize(eval("+string(f)+"))");
```

D.2.3.17 which

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: which(command); command = string expression

Return: Absolute pathname of command, if found in search path. Empty string, otherwise.

Note: Based on the Unix command 'which'.

Example:

```
LIB "general.lib";
which("sh");
```

D.2.3.18 primecoeffs

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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 <= min(p,32003) 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

Note: the procedure works for small integers only, just by testing all primes (not to be considered as serious prime factorization!)

Example:

```
LIB "general.lib";
primecoeffs(intvec(7*8*121,7*8));"";
ring r = 0,(b,c,t),dp;
ideal I = -13b6c3t+4b5c4t,-10b4c2t-5b4ct2;
primecoeffs(I);
```

D.2.3.19 timeStd

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: timeStd(i,d), i ideal, d integer

Return: std(i) if the standard basis computation finished after d-1 seconds and i otherwhise

Example:

```
LIB "general.lib";
ring r=32003,(a,b,c,d,e),dp;
int n=6;
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;
timeStd(i,2);
timeStd(i,20);
```

D.2.3.20 timeFactorize

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

Usage: timeFactorize(p,d); poly p , integer d

Return: factorize(p) if the factorization finished after d-1
seconds otherwhise 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);
//timeFactorize(p,20);
```

D.2.3.21 factorH

Procedure from library `general.lib` (see Section D.2.3 [general_lib], page 537).

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
system("random",992851262);
//factorize(p);  //slow
system("random",992851262);
factorH(p);
```

D.2.4 inout_lib

Library: inout.lib

Purpose: Printing and Manipulating In- and Output

Procedures:

D.2.4.1 allprint

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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‘ is already global
allprint("M =",M);
↦ M =
↦ 1,0,0,
```

```
↦ 0,1,0,
↦ 0,0,0
kill ALLprint;
```

D.2.4.2 lprint

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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.2.4.3 pmat

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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.2.4.4 rMacaulay

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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 assinged 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.2.4.5 show

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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: (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);
↦ // qring: (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
map F=r,a2,b^2,3*c3;
show(F);
↦ // i-th variable of preimage ring is mapped to @map[i]
↦ // @map                  [1]  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.2.4.6 showrecursive

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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 occuring 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.2.4.7 split

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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.2.4.8 tab

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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.2.4.9 pause

Procedure from library `inout.lib` (see Section D.2.4 [inout_lib], page 545).

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 249; Section 5.1.114 [read], page 206.

D.2.5 poly_lib

Library: poly.lib

Purpose: Procedures for Manipulating Polys, Ideals, Modules

Authors: O. Bachmann, G.-M. Greuel, A. Fruehbis

Procedures:

D.2.5.1 cyclic

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

Usage: cyclic(n); n integer

Return: ideal of cyclic n-roots from 1-st n variables of basering

Example:

```
LIB "poly.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.2.5.2 elemSymmId

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

Return: ideal of elementary symmetric polynomials for 1-st n variables of basering

Example:

```
LIB "poly.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.2.5.3 katsura

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.lib";
ring r; basering;
↦ //   characteristic : 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)
↦ //   characteristic : 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.2.5.4 freerank

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.5 is_zero

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.6 lcm

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.7 maxcoef

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.8 maxdeg

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.9 [maxdeg1], page 555.

D.2.5.9 maxdeg1

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.10 mindeg

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.11 [mindeg1], page 556.

D.2.5.11 mindeg1

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.12 normalize

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.13 rad_con

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.14 content

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

Usage: content(f); f polynomial/vector

Return: number, the content (greatest common factor of coefficients) of the polynomial/vector f

Example:

```
LIB "poly.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 5.1.8 [cleardenom], page 133.

D.2.5.15 numerator

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

Usage: numerator(n); n number

Return: number, the numerator of n

Example:

```
LIB "poly.lib";
ring r = 0,x, dp;
number n = 3/2;
numerator(n);
↦ 3
```

See also: Section 5.1.8 [cleardenom], page 133; Section D.2.5.14 [content], page 557; Section D.2.5.16 [denominator], page 557.

D.2.5.16 denominator

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

Usage: denominator(n); n number

Return: number, the denominator of n

Example:

```
LIB "poly.lib";
ring r = 0,x, dp;
number n = 3/2;
denominator(n);
↦ 2
```

See also: Section 5.1.8 [cleardenom], page 133; Section D.2.5.14 [content], page 557; Section D.2.5.15 [numerator], page 557.

D.2.5.17 mod2id

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.18 id2mod

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.19 substitute

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.20 subrInterred

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.21 newtonDiag

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.5.22 hilbPoly

Procedure from library `poly.lib` (see Section D.2.5 [poly_lib], page 551).

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 "poly.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.2.6 random_lib

Library: random.lib

Purpose: Creating Random and Sparse Matrices, Ideals, Polys

Procedures:

D.2.6.1 genericid

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.2 randomid

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.3 randommat

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.4 sparseid

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.5 sparsematrix

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.6 sparsemat

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.7 sparsepoly

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.8 sparsetriag

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.9 sparseHomogIdeal

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.10 triagmatrix

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.11 randomLast

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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.2.6.12 randomBinomial

Procedure from library `random.lib` (see Section D.2.6 [random_lib], page 561).

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 choosen 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.2.7 redcgs_lib

Library: redcgs.lib

Purpose: Reduced Comprehensive Groebner Systems.

Purpose: 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 rea-

sonable 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 528.

D.2.7.1 setglobalrings

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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;
↦ //   characteristic : 0
↦ //   2 parameter    : a b
↦ //   minpoly        : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
@P;
↦ //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering lp
↦ //                  : names    a b
@RP;
↦ //   characteristic : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering lp
↦ //                  : names    a b
```

D.2.7.2 memberpos

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.3 subset

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.4 pdivi

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

Usage: pdivi(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=pdivi(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.2.7.5 facvar

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.6 redspec

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.7 pnormalform

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.8 buildtree

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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 remaning 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 readed 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.2.7.9 buildtreetoMaple

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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.2.7.10 finalcases

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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-a
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.2.7.11 mrcgs

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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 substraction. 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 **
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
↦ // ** redefining @P
↦ // ** redefining @RP
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 **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ [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.2.7.12 rcgs

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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 substraction. 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
↦ // ** redefining @P
↦ // ** redefining @RP
↦ // ** redefining @R
↦ // ** redefining @P
↦ // ** redefining @RP
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 **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ [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.2.7.13 crcgs

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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 substraction. 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
↦ // ** redefining @R
↦ // ** redefining @P
↦ // ** redefining @RP
↦ // ** NP2 is no standard basis
↦ // ** NP2 is no standard basis
↦ // ** redefining @R
↦ // ** redefining @P
↦ // ** redefining @RP
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 **
```

```
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ [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.2.7.14 cantodiffcgs

Procedure from library `redcgs.lib` (see Section D.2.7 [redcgs_lib], page 567).

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
```

```
↦ // ** redefining @R
↦ // ** redefining @P
↦ // ** redefining @RP
↦ // ** NP2 is no standard basis
↦ // ** NP2 is no standard basis
↦ // ** redefining @R
↦ // ** redefining @P
↦ // ** redefining @RP
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 **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ // ** redefining NP **
↦ // ** redefining MP **
↦ [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.2.8 ring_lib

Library: ring.lib

Purpose: Manipulating Rings and Maps

Procedures:

D.2.8.1 changechar

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

Usage: changechar(c[,r]); c=string, r=ring

Return: ring R, obtained from the ring r [default: r=basering], by changing charstr(r) to c.

Note: Works for qrings if map from old_char to new_char is implemented. This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
ring r=0,(x,y,u,v),(dp(2),ds);
def R=changechar("2,A"); R;"";
↦ //   characteristic : 2
↦ //   1 parameter    : A
↦ //   minpoly        : 0
↦ //   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;
↦ //   characteristic : 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.2.8.2 changeord

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

Usage: changeord(newordstr[,r]); newordstr=string, r=ring/qring

Return: ring R, obtained from the ring r [default: r=basering], by changing ordstr(r) to newordstr. If, say, newordstr=("R","wp(2,3),dp") and if the ring r exists and has >=3 variables, the ring R will be equipped with the monomial ordering wp(2,3),dp.

Note: This proc uses 'execute' or calls a procedure using 'execute'.

Example:

```
LIB "ring.lib";
ring r=0,(x,y,u,v),(dp(2),ds);
def R=changeord("wp(2,3),dp"); R; "";
↦ //   characteristic : 0
↦ //   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("lp",Q); setring Q'; Q';
↦ //   characteristic : 0
↦ //   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.2.8.3 changevar

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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; "";
↦ //   characteristic : 0
↦ //   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';
↦ //   characteristic : 0
↦ //   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.2.8.4 defring

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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;"";
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 2
↦ //   1 parameter    : A
↦ //   minpoly        : 0
↦ //   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
kill R,r;
```

D.2.8.5 defrings

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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; "";
↦ //   characteristic : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering ds
↦ //                  : names    x y z a b
↦ //        block   2 : ordering C
↦
def S30=defrings(30); S30;
↦ //   characteristic : 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.2.8.6 defringp

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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; "";
↦ //   characteristic : 0
↦ //   number of vars : 5
↦ //        block   1 : ordering dp
↦ //                  : names    x y z a b
↦ //        block   2 : ordering C
```

```
↦
def P30=defringp(30); P30;
↦ //   characteristic : 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.2.8.7 extendring

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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: (0),(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: (0),(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: (0),(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: (0),(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: (0),(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.2.8.8 fetchall

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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 inside a procedure.
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.2.8.9 [imapall], page 610.

D.2.8.9 imapall

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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 inside a procedure
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.2.8.8 [fetchall], page 609.

D.2.8.10 mapall

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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 inside a procedure.

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.2.8.11 ord_test

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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
```

D.2.8.12 ringtensor

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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                    [0]  ring
↦ //   characteristic : 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                   [0]  *qring
```

```
↦ //   characteristic : 0
↦ //   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 4.18.3 [ring operations], page 120.

D.2.8.13 ringweights

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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.2.8.14 preimageLoc

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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);
map psi=S,x,y,z;
ideal null;
setring S;
ideal nu=preimageLoc("R","psi","null");
nu;
↦ nu[1]=x
```

See also: Section 5.1.104 [preimage], page 198.

D.2.8.15 rootofUnity

Procedure from library `ring.lib` (see Section D.2.8 [ring_lib], page 604).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : q
↦ //   minpoly        : (q4+1)
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x y z
↦ //        block   2 : ordering C
```

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 614).

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 614).

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 614).

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 614).

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 614).

Usage: flatten(A); A matrix

Return: ideal, generated by all entries from A

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
```

D.3.1.6 genericmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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 614).

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 614).

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 614).

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 614).

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 614).

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 614).

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 tensor

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

Usage: tensor(A,B); A,B matrices

Return: matrix, tensor product of A and B

Example:

```
LIB "matrix.lib";
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
```

D.3.1.14 unitmat

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.15 gauss_col

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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,  (1/2b)*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.3.1.24 [colred], page 629.

D.3.1.16 gauss_row

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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,   (1/2b)*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.3.1.23 [rowred], page 626.

D.3.1.17 addcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.18 addrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.19 multcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.20 multrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.21 permcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.22 permrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.23 rowred

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
```

```
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, (2a)
↦ 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);
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
```

```
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, (2a)
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.3.1.16 [gauss_row], page 623.

D.3.1.24 colred

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
```

```
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, (2a)
↦ 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);
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 2, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
```

```
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, 0,    0,
↦ 0, 0, 0, 0, 0, 0, (2b), 0,
↦ 0, 0, 0, 0, 0, 0, 0,    (2a)
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, 0,
↦ 0, 0, 0, 0, 0, 0, 0, (2a)
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.3.1.15 [gauss_col], page 622.

D.3.1.25 linear_relations

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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(M*REL);
↦ 0,
↦ 0
```

D.3.1.26 rm_unitrow

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.27 rm_unitcol

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.28 headStand

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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.3.1.29 symmetricBasis

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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:
↦ //   characteristic : 0
↦ //   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.3.1.30 [exteriorBasis], page 635.

D.3.1.30 exteriorBasis

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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:
↦ //   characteristic : 0
↦ //   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.3.1.29 [symmetricBasis], page 634.

D.3.1.31 symmetricPower

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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.3.1.32 [exteriorPower], page 637.

D.3.1.32 exteriorPower

Procedure from library `matrix.lib` (see Section D.3.1 [matrix_lib], page 614).

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:";
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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.3.1.31 [symmetricPower], page 635.

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 638).

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 639; Section D.3.2.3 [inverse_L], page 640.

D.3.2.2 inverse_B

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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 638; Section D.3.2.3 [inverse_L], page 640.

D.3.2.3 inverse_L

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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 638; Section D.3.2.2 [inverse_B], page 639.

D.3.2.4 sym_gauss

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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 638).

Usage: orthogonalize(A); A = matrix of constants

Return: matrix, orthogonal basis of the colum 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 638).

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 638).

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 638).

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 diplayed 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 638).

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 638).

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 638).

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 638).

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 638).

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 638).

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 638).

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 638).

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 638).

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
```

D.3.2.18 eigenvals

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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
```

D.3.2.19 minipoly

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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 638).

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] and 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 638).

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.5.7 [gmssing_lib], page 919; Section D.3.2.20 [spnf], page 649.

D.3.2.22 jordan

Procedure from library `linalg.lib` (see Section D.3.2 [linalg_lib], page 638).

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 638).

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 638).

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 638).

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

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. See [Cheze and Lecerf, Lifting and recombination techniques for absolute factorization].

Procedures: See also: Section 5.1.30 [factorize], page 147.

D.4.1.1 absFactorize

Procedure from library `absfact.lib` (see Section D.4.1 [absfact_lib], page 652).

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 `j`th (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.4.18.15 [absPrimdecGTZ], page 787; Section 5.1.30 [factorize], page 147.

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: Auxiliary procedures:

D.4.2.1 algebra_containment

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 653).

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 653).

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 653).

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 653).

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 choosen. 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 653).

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 653).

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 differnt algorithms are used depending on c = 1,2,3. If c is not given or c=0, a heuristically best method is choosen. 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 653).

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 653).

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 interprete 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 653).

Usage: noetherNormal(id[,p]); id ideal, p integer

Return:

a list l of two ideals, say I,J:
- I is generated by a subset of the variables with size(I) = dim(id)
- J defines a map (coordinate change in the basering), such that:
  if we define map phi=basering,J;
  then k[var(1),...,var(n)]/phi(id) is finite over k[I].

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 653).

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 661.

D.4.2.11 finitenessTest

Procedure from library `algebra.lib` (see Section D.4.2 [algebra_lib], page 653).

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 653).

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 assprime_lib

Library: assPrimes.lib

Purpose: associated primes of a zero-dimensional ideal

Authors: N. Idrees nazeranjawwad@gmail.com
G. Pfister pfister@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.4.18 [primdec_lib], page 779.

D.4.3.1 assPrimes

Procedure from library `assprime.lib` (see Section D.4.3 [assprime_lib], page 663).

Usage: assPrimes(I,[n],[a]); I ideal or module,
optional: n number of processors (for parallel computing), a - a=1: method of Eisenbud/Hunecke/Vasconcelos
- a=2: method of Gianni/Trager/Zacharias
- a=3: mathod of Monico
assPrimes(I) chooses n=a=1

Assume: I is zero-dimensional over Q[variables]

Return: a list pr of associated primes of I:

Example:

```
LIB "assprime.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]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=27a+41b+75c+100d-24e+f
↦ [2]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
```

```
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-46008a-6\
   9864b-115416c-183900d-184932e-270732f+8521
↦ [3]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-810a-123\
   0b+10134c-16500d-225108e-269058f+20653
↦ [4]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-50382a-7\
   6506b-127566c-200100d-181044e-270894f+8161
↦ [5]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-72090a-1\
   09470b-187866c-280500d-161748e-271698f+34513
↦ [6]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2+6966a+10578b+234\
   78c+21300d-81468e-89418f+7927
↦ [7]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
```

```
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-51354a-7\
   7982b-130266c-203700d-180180e-270930f+20497
↦ [8]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=688ac-750ad-3050bd+9271d2-12546ae-16968be-5058ce-7014de-1105e2-14\
   946af-1376bf-2064cf-3162df-6198ef-5624f2-702a-1066b-1606c-2975d-5649e-749\
   9f+817
↦ [9]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+51354a+7\
   7982b+155034c+176700d-271476e-267126f+71851
↦ [10]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+50382a+7\
   6506b+152334c+173100d-270612e-267162f+58543
↦ [11]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2-6966a-10578b-152\
   22c-30300d-69084e-89934f+961
↦ [12]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=6192ac-6750ad-27450bd+83439d2-112914ae-152712be-45522ce-63126de-9\
   945e2-134514af-12384bf-18576cf-28458df-55782ef-50616f2-11502a-17466b-2885\
   4c-45975d-46233e-67683f+8281
```

```
↦ [13]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2-6642a-10086b-143\
   22c-29100d-69372e-89922f+4957
↦ [14]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2-8424a-12792b-192\
   72c-35700d-67788e-89988f+1603
↦ [15]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+810a+123\
   0b+14634c-10500d-226548e-268998f+21463
↦ [16]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+72090a+1\
   09470b+212634c+253500d-289908e-266358f+106603
↦ [17]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-162a-246\
   b+11934c-14100d-225684e-269034f+9091
↦ [18]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
```

```
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+71118a+1\
   07994b+209934c+249900d-289044e-266394f+92911
↦ [19]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+162a+246\
   b+12834c-12900d-225972e-269022f+9253
↦ [20]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2+46008a+6\
   9864b+140184c+156900d-266724e-267324f+54529
↦ [21]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2+6642a+10086b+225\
   78c+20100d-81180e-89430f+11599
↦ [22]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=688ac-750ad-3050bd+9271d2-12546ae-16968be-5058ce-7014de-1105e2-14\
   946af-1376bf-2064cf-3162df-6198ef-5624f2+702a+1066b+2294c+2225d-6897e-744\
   7f+1519
↦ [23]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
```

```
↦    _[6]=2cd+2be+2af+f
↦    _[7]=8256ac-9000ad-36600bd+111252d2-150552ae-203616be-60696ce-84168de-\
   13260e2-179352af-16512bf-24768cf-37944df-74376ef-67488f2+8424a+12792b+275\
   28c+26700d-82764e-89364f+10027
↦ [24]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=24768ac-27000ad-109800bd+333756d2-451656ae-610848be-182088ce-2525\
   04de-39780e2-538056af-49536bf-74304cf-113832df-223128ef-202464f2-71118a-1\
   07994b-185166c-276900d-162612e-271662f+21793
↦ [25]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=6192ac-6750ad-27450bd+83439d2-112914ae-152712be-45522ce-63126de-9\
   945e2-134514af-12384bf-18576cf-28458df-55782ef-50616f2+11502a+17466b+3504\
   6c+39225d-66681e-66831f+19783
↦ [26]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=162a+246b+450c+600d-144e+6f-155
↦ [27]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=162a+246b+450c+600d-144e+6f+1
↦ [28]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=27a+41b+75c+100d-24e+f+26
↦ [29]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
```

```
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=81a+123b+225c+300d-72e+3f-32
↦ [30]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=54a+82b+150c+200d-48e+2f-73
↦ [31]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=27a+41b+75c+100d-24e+f+1
↦ [32]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=162a+246b+450c+600d-144e+6f+317
↦ [33]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=81a+123b+225c+300d-72e+3f-29
↦ [34]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=81a+123b+225c+300d-72e+3f+110
↦ [35]:
↦    _[1]=a2+d2+2ce+2bf+a
↦    _[2]=2ab+2de+2cf+b
↦    _[3]=b2+2ac+e2+2df+c
↦    _[4]=2bc+2ad+2ef+d
↦    _[5]=c2+2bd+2ae+f2+e
↦    _[6]=2cd+2be+2af+f
↦    _[7]=27a+41b+75c+100d-24e+f+27
↦ [36]:
↦    _[1]=a2+d2+2ce+2bf+a
```

```
↦     _[2]=2ab+2de+2cf+b
↦     _[3]=b2+2ac+e2+2df+c
↦     _[4]=2bc+2ad+2ef+d
↦     _[5]=c2+2bd+2ae+f2+e
↦     _[6]=2cd+2be+2af+f
↦     _[7]=162a+246b+450c+600d-144e+6f+161
↦  [37]:
↦     _[1]=a2+d2+2ce+2bf+a
↦     _[2]=2ab+2de+2cf+b
↦     _[3]=b2+2ac+e2+2df+c
↦     _[4]=2bc+2ad+2ef+d
↦     _[5]=c2+2bd+2ae+f2+e
↦     _[6]=2cd+2be+2af+f
↦     _[7]=54a+82b+150c+200d-48e+2f+127
↦  [38]:
↦     _[1]=a2+d2+2ce+2bf+a
↦     _[2]=2ab+2de+2cf+b
↦     _[3]=b2+2ac+e2+2df+c
↦     _[4]=2bc+2ad+2ef+d
↦     _[5]=c2+2bd+2ae+f2+e
↦     _[6]=2cd+2be+2af+f
↦     _[7]=162a+246b+450c+600d-144e+6f+97
↦  [39]:
↦     _[1]=a2+d2+2ce+2bf+a
↦     _[2]=2ab+2de+2cf+b
↦     _[3]=b2+2ac+e2+2df+c
↦     _[4]=2bc+2ad+2ef+d
↦     _[5]=c2+2bd+2ae+f2+e
↦     _[6]=2cd+2be+2af+f
↦     _[7]=162a+246b+450c+600d-144e+6f+65
↦  [40]:
↦     _[1]=a2+d2+2ce+2bf+a
↦     _[2]=2ab+2de+2cf+b
↦     _[3]=b2+2ac+e2+2df+c
↦     _[4]=2bc+2ad+2ef+d
↦     _[5]=c2+2bd+2ae+f2+e
↦     _[6]=2cd+2be+2af+f
↦     _[7]=81a+123b+225c+300d-72e+3f+113
```

D.4.3.2 zeroR

Procedure from library `assprime.lib` (see Section D.4.3 [assprime_lib], page 663).

Usage: zeroR(I,[n]); I ideal, optional: n number of processors (for parallel computing)

Assume: I is zero-dimensional in Q[variables]

Return: the radical of I

Example:

```
LIB "assprime.lib";
ring R = 0, (x,y), dp;
ideal I = xy4-2xy2+x, x2-x, y4-2y2+1;
zeroR(I);
↦ _[1]=y2-1
```

```
↦ _[2]=x2-x
```

D.4.4 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 516.

D.4.4.1 BelongSemig

Procedure from library `cimonom.lib` (see Section D.4.4 [cimonom_lib], page 671).

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.4.4.2 MinMult

Procedure from library `cimonom.lib` (see Section D.4.4 [cimonom_lib], page 671).

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.4.4.3 CompInt

Procedure from library `cimonom.lib` (see Section D.4.4 [cimonom_lib], page 671).

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.4.5 elim_lib

Library: elim.lib

Purpose: Elimination, Saturation and Blowing up

Procedures:

D.4.5.1 blowup0

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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 tranform 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
```

```
↦ sT[2]=x(1)*y(2)-x(2)*y(1)
↦ sT[3]=x(2)^2*y(2)+x(1)*y(1)
↦ sT[4]=x(2)^3+x(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
↦ //   characteristic : 0
↦ //   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(2)-y(3)
↦ sT[2]=x(1)-x(3)
↦ sT[3]=y(1)+x(3)^2*y(3)+x(3)^4*y(3)
↦ sT[4]=x(2)^2+x(3)^3+x(3)^5
eD;                         //ideal of exceptional divisor
↦ eD[1]=x(1)
↦ eD[2]=x(3)
↦ eD[3]=x(2)^2
//Note that the different affine charts are {y(i)=1}
```

See also: Section D.4.23.1 [blowUp], page 843; Section D.4.23.2 [blowUp2], page 844.

D.4.5.2 elimRing

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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/moduel 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]:
↦    //   characteristic : 0
↦ //   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]:
↦    //   characteristic : 0
↦ //   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;
```

```
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : (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.4.5.3 elim

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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. Works also 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 choosen method.
If the variables in the basering have weights these weights are used in elimRing. If a string "withWeigts" as (optional) argument is given @sc{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:
↦ //   characteristic : 0
↦ //   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;
qring T = std(ideal(ax+y2+v3,(x+v)^2));
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, 6 generator(s)
↦ [0,(a)*z-u3]
↦ [(a)*z-u3]
↦ [0,(-a)*u-v3+2*u3v+u2v2]
↦ [(-a)*u-v3+2*u3v+u2v2]
↦ [0,v]
↦ [v]
```

See also: Section D.4.5.4 [elim1], page 677; Section 5.1.23 [eliminate], page 143.

D.4.5.4 elim1

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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.
This proc uses 'execute' or calls a procedure using 'execute'.

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.4.5.3 [elim], page 676; Section 5.1.23 [eliminate], page 143.

D.4.5.5 elim2

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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 This proc uses 'execute' or calls a procedure using 'execute'.

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.4.5.3 [elim], page 676; Section D.4.5.4 [elim1], page 677; Section 5.1.23 [eliminate], page 143.

D.4.5.6 nselect

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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.4.5.8 [select], page 680; Section D.4.5.9 [select1], page 680.

D.4.5.7 sat

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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

Display: saturation exponent during computation if printlevel >=1

Example:

```
LIB "elim.lib";
int p      = printlevel;
ring r     = 2,(x,y,z),dp;
poly F     = x5+y5+(x-y)^2*xyz;
ideal j    = jacob(F);
sat(j,maxideal(1));
↦ [1]:
↦    _[1]=x3+x2y+xy2+y3
↦    _[2]=y4+x2yz+y3z
↦    _[3]=x2y2+y4
↦ [2]:
↦    4
printlevel = 2;
sat(j,maxideal(2));
↦ // compute quotient 1
↦ // compute quotient 2
↦ // compute quotient 3
↦ // saturation becomes stable after 2 iteration(s)
↦
↦ [1]:
↦    _[1]=x3+x2y+xy2+y3
```

```
↦     _[2]=y4+x2yz+y3z
↦     _[3]=x2y2+y4
↦  [2]:
↦     2
printlevel = p;
```

D.4.5.8 select

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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.4.5.6 [nselect], page 678; Section D.4.5.9 [select1], page 680.

D.4.5.9 select1

Procedure from library `elim.lib` (see Section D.4.5 [elim_lib], page 672).

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.4.5.6 [nselect], page 678; Section D.4.5.8 [select], page 680.

D.4.6 grwalk_lib

Library: grwalk.lib

Purpose: Groebner Walk Conversion Algorithms

Author: I Made Sulandra

Procedures:

D.4.6.1 fwalk

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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.4.6.3 [awalk1], page 682; Section D.4.6.4 [awalk2], page 683; Section 5.1.44 [groebner], page 155; Section D.4.6.6 [gwalk], page 684; Section D.4.6.5 [pwalk], page 683; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section D.4.6.2 [twalk], page 682.

D.4.6.2 twalk

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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.4.6.3 [awalk1], page 682; Section D.4.6.4 [awalk2], page 683; Section D.4.6.1 [fwalk], page 681; Section 5.1.44 [groebner], page 155; Section D.4.6.6 [gwalk], page 684; Section D.4.6.5 [pwalk], page 683; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224.

D.4.6.3 awalk1

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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.4.6.4 [awalk2], page 683; Section D.4.6.1 [fwalk], page 681; Section 5.1.44 [groebner], page 155; Section D.4.6.6 [gwalk], page 684; Section D.4.6.5 [pwalk], page 683; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section D.4.6.2 [twalk], page 682.

D.4.6.4 awalk2

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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.4.6.3 [awalk1], page 682; Section D.4.6.1 [fwalk], page 681; Section 5.1.44 [groebner], page 155; Section D.4.6.6 [gwalk], page 684; Section D.4.6.5 [pwalk], page 683; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section D.4.6.2 [twalk], page 682.

D.4.6.5 pwalk

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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,2,3);
↦ _[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.4.6.3 [awalk1], page 682; Section D.4.6.4 [awalk2], page 683; Section D.4.6.1 [fwalk], page 681; Section 5.1.44 [groebner], page 155; Section D.4.6.6 [gwalk], page 684; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section D.4.6.2 [twalk], page 682.

D.4.6.6 gwalk

Procedure from library `grwalk.lib` (see Section D.4.6 [grwalk_lib], page 681).

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 = y3+xyz+y2z+xz3, 3+xy+x2y+y2z;
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.4.6.3 [awalk1], page 682; Section D.4.6.4 [awalk2], page 683; Section D.4.6.1 [fwalk], page 681; Section 5.1.44 [groebner], page 155; Section D.4.6.5 [pwalk], page 683; Section 5.1.133 [std], page 223; Section 5.1.134 [stdfglm], page 224; Section D.4.6.2 [twalk], page 682.

D.4.7 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.4.7.1 canonMap

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

Usage: canonMap(id); id= ideal/module,

Return: a list L, the kernel in two different representations and
the cokernel of the canonical map
M —> Extˆc_R(Extˆc_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=Rˆn/id
c is the codimension of M
L[1] is the preimage of the kernel in R resp. Rˆn
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(2)-w*gen(1)
↦    _[2]=z*gen(4)-w*gen(3)
↦    _[3]=x*gen(1)-y*gen(3)
↦    _[4]=x*gen(2)-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.4.7.2 cup

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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));
↦ 0,1,0, 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,
↦ 0,0,0, -1,0,0,0,0,0,1,0, 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));
↦ 0,1,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,0,0,1,0,0,0,0,0,0
// 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 (Extˆ1) = 2
↦ // kbase of Extˆ1(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 Extˆ1:
↦ //  - 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 (Extˆ2) = 2
↦ // kbase of Extˆ2(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 Extˆ2)
↦ 0,-1,0, 0,y,
↦ 0,0, -y,y,0,
↦ 0,0, -1,1,0,
↦ 0,-1,0, 0,y
↦ ////// end level 2 //////
↦ // the associated matrices of the bilinear mapping 'cup'
↦ // corresponding to the kbase elements of Extˆ2(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:
↦ 0,1,
↦ 0,0
↦ //-----component 2:
↦ 0, 0,
↦ -1,1
↦ ////// end level 3 //////
↦ 0,1,0, 0,0,
↦ 0,0,-1,1,0
printlevel = p;
```

D.4.7.3 cupproduct

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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 choosen 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));
↦ 0,1,0, 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,
↦ 0,0,0, -1,0,0,0,0,0,1,0, 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 Extˆp+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 Extˆp(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 Extˆp+q)
↦ 0,0, 0, -1, 0,   0, 0, 0,   y,   1,  0,  0,  0,  0,   y,   0,  0,
↦ 0,0, 0, 0,  y,   0, 0, y,   0,   0,  -y, 0,  0,  y,   0,   0,  0,
↦ 0,1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ 0,0, y, 0,  0,   -y,0, 0,   0,   0,  0,  0,  x2y,0,   0,   x2y,0,
↦ 0,0, 1, 0,  0,   -1,0, 0,   0,   0,  0,  0,  x2, 0,   0,   x2, 0,
↦ 0,1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ 0,0, 0, 0,  -x2, 0, 0, -x2, 0,   0,  x2, 0,  0,  -x2, 0,   0,  0,
↦ 0,0, 0, x2, 0,   0, 0, 0,   -x2y,-x2,0,  0,  0,  0,   -x2y,0,  0,
↦ 0,0, 0, 0,  -1,  0, 0, -1,  0,   0,  1,  0,  0,  -1,  0,   0,  0,
↦ 0,0, 0, -1, 0,   0, 0, 0,   y,   1,  0,  0,  0,  0,   y,   0,  0,
↦ 0,0, -1,0,  0,   1, 0, 0,   0,   0,  0,  0,  -x2,0,   0,   -x2,0,
↦ 0,1, 0, 0,  0,   0, y, 0,   0,   0,  0,  x2, 0,  0,   0,   0,  -x2y,
↦ 0,-1,0, 0,  0,   0, -y,0,   0,   0,  0,  -x2,0,  0,   0,   0,  x2y,
```

```
↦ 0,0, y, 0,  0,   -y,0, 0,   0,   0,  0,  0,  x2y,0,   0,   x2y,0,
↦ 0,0, 0, -x2,0,   0, 0, 0,   x2y, x2, 0,  0,  0,  0,   x2y, 0,  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:
↦ 0,1,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0,
↦ 0,0,0,0
↦ //----component 2:
↦ 0,0,-1,0,
↦ 0,1,0, 0,
↦ 0,0,0, 0,
↦ 0,0,0, 0
↦ //----component 3:
↦ 0,0,0,-1,
↦ 0,0,0,0,
↦ 0,1,0,0,
↦ 0,0,0,0
↦ //----component 4:
↦ 0,0,0, 0,
↦ 1,0,0, 1,
↦ 0,0,-1,0,
↦ 0,1,0, 0
↦ ////// end level 3 //////
show(l[1]);show(l[2]);
↦ // matrix, 4x17
↦ 0,1,0, 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,
↦ 0,0,0, -1,0,0,0,0,0,1,0, 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.4.7.4 depth

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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 maxideal in the basering (which is then assumed to be local)
- if called with 2 arguments: the depth of M'=coker(M) w.r.t. the ideal I.

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
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.4.7.5 Ext_R

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.6 Ext

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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 Extˆk(M',N') if v is of type int
- a list of Extˆk (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 Extˆk
l[2] = SB of Ext/list of SB of Extˆk
l[3] = matrix/list of matrices, each representing a kbase of Extˆk (if finite dimensional)

Display: printlevel >=0: dimension, vdim of Extˆk for each k (default).
printlevel >=1: matrices Ak, Ak+1 and kbase of Extˆk in Hom(Fk,G0) (if finite dimensional)

Note: In order to compute Extˆk(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);    // Extˆk(r/i,r/j) for k=0,1,2 over r
↦ // Computing Extˆ0 (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 Extˆ0:  -1
↦
↦ // Computing Extˆ1 (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 Extˆ1:  0
↦ // vdim of Extˆ1:       10
↦
↦ // Computing Extˆ2 (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 Extˆ2:  -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.4.7.7 fitting

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.8 flatteningStrat

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.9 Hom

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.10 homology

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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);
qring qr=std(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.4.7.11 isCM

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.12 isFlat

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.13 isLocallyFree

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.14 isReg

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.15 hom_kernel

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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:

```
ker(A') --->  M' --A'--> N'
   |^          |^          |^
   |           |           |
   R^r  ---> R^m --A--> R^n
   |^          |^          |^
   |K          |M          |N
   |           |           |
   R^s  ---> R^p -----> R^q
```

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.4.7.16 kohom

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.17 kontrahom

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.18 KoszulHomology

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.19 tensorMod

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.7.20 Tor

Procedure from library `homolog.lib` (see Section D.4.7 [homolog_lib], page 684).

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.4.8 intprog_lib

Library: intprog.lib

Purpose: Integer Programming with Groebner Basis Methods

Author: Christine Theis, email: ctheis@math.uni-sb.de

Procedures:

D.4.8.1 solve_IP

Procedure from library `intprog.lib` (see Section D.4.8 [intprog_lib], page 708).

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 513.

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 516; Section D.4.8 [intprog_lib], page 708; Section D.4.28 [toric_lib], page 880.

D.4.9 lll_lib

Library: LLL.lib

Purpose: Integral LLL-Algorithm

Author: Alberto Vigneron-Tenorio and Alfredo Sanchez-Navarro
email: alberto.vigneron@uca.es, alfredo.sanchez@uca.es

Procedures:

D.4.9.1 LLL

Procedure from library `lll.lib` (see Section D.4.9 [lll_lib], page 710).

Example:

```
LIB "lll.lib";
list l=list(1,-2,4),list(2,1,-7);
LLL(l);
↦ [1]:
↦    [1]:
↦       1
↦    [2]:
↦       -2
↦    [3]:
↦       4
↦ [2]:
↦    [1]:
↦       3
↦    [2]:
↦       -1
↦    [3]:
↦       -3
```

D.4.10 modstd_lib

Library: modstd.lib

Purpose: Groebner basis of ideals

Authors: A. Hashemi Amir.Hashemi@lip6.fr
G. Pfister pfister@mathematik.uni-kl.de
H. Schoenemann hannes@mathematik.uni-kl.de
S. 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 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:

D.4.10.1 modStd

Procedure from library `modstd.lib` (see Section D.4.10 [modstd_lib], page 710).

Usage: modStd(I); I ideal

Assume: 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: a standard basis of I if no warning appears;

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular methods. If a warning appears then the result is a standard basis containing I and with high probability a standard basis of I.
By default the procedure computes a standard basis of I with high probability, but if the optional parameter #[2] = 1, it is exact. 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 and exactness = 0,
- variant = 2, if I is not homogeneous, 1-block-ordering and exactness = 0,
- variant = 3, if I is not homogeneous, complicated ordering (lp or > 1 block) and exactness = 0,
- variant = 4, if I is homogeneous and exactness = 1,
- variant = 5, if I is not homogeneous and exactness = 1.

Example:

```
LIB "modstd.lib";
ring r = 0,(x,y,z,t),dp;
ideal I = 3x3+x2+1, 11y5+y3+2, 5z4+z2+4;
ideal J = modStd(I);
↦ ==========================================================================
↦ WARNING: Ideal generated by output may be greater than input ideal.
↦ ==========================================================================
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);
↦ ========================================================================
↦ WARNING: Ideal generated by output may be greater than input ideal.
↦ ========================================================================
J;
↦ J[1]=x3+1/3x2t+1/3t3
↦ J[2]=z4+1/5z2t2+4/5t4
↦ J[3]=y5+1/11y3t2+2/11t5
ring s = 0,(x,y,z),ds;
ideal I = jacob(x5+y6+z7+xyz);
ideal J1 = modStd(I,1,1);
J1;
↦ J1[1]=xy+7z6
↦ J1[2]=xz+6y5
↦ J1[3]=yz+5x4
↦ J1[4]=x5-6/5y6
↦ J1[5]=y6-7/6z7
↦ J1[6]=z8+30/7x4y5
// requires MP
//ideal J2 = modStd(I,3);
//J2;
//size(reduce(J1,J2));
//size(reduce(J2,J1));
ring rr = 0,x(1..4),lp;
ideal I = cyclic(4);
ideal J1 = modStd(I,1);
↦ ========================================================================
↦ WARNING: Ideal generated by output may be greater than input ideal.
↦ ========================================================================
// requires MP
//ideal J2 = modStd(I,2,1);
//size(reduce(J1,J2));
//size(reduce(J2,J1));
```

D.4.10.2 modHenselStd

Procedure from library `modstd.lib` (see Section D.4.10 [modstd_lib], page 710).

Usage: modHenselStd(I);

Return: a standard basis of I;

Note: The procedure computes a standard basis of I (over the rational numbers) by using modular computations and Hensellifting. For further experiments see procedure modS.

Example:

```
LIB "modstd.lib";
ring r = 0,(x,y,z),dp;
ideal I = 3x3+x2+1,11y5+y3+2,5z4+z2+4;
ideal J = modHenselStd(I);
J;
↦ J[1]=3x3+x2+1
↦ J[2]=5z4+z2+4
↦ J[3]=11y5+y3+2
```

D.4.10.3 modS

Procedure from library `modstd.lib` (see Section D.4.10 [modstd_lib], page 710).

Usage: modS(I,L); I ideal, L intvec of primes
if size(#)>0 std is used instead of groebner

Return: an ideal which is with high probability a standard basis

Note: This procedure is designed for fast experiments.
It is not tested whether the result is a standard basis. It is not tested whether the result generates I.

Example:

```
LIB "modstd.lib";
list L = 3,5,11,13,181,32003;
ring r = 0,(x,y,z,t),dp;
ideal I = 3x3+x2+1,11y5+y3+2,5z4+z2+4;
I = homog(I,t);
ideal J = modS(I,L);
J;
↦ J[1]=x3+1/3x2t+1/3t3
↦ J[2]=z4+1/5z2t2+4/5t4
↦ J[3]=y5+1/11y3t2+2/11t5
```

D.4.11 monomial_lib

Library: monomial.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.4.11.1 isMonomial

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

Usage: isMonomial (I); I ideal.

Return: 1, if I is monomial ideal; 0, otherwise.

Assume: I is an ideal of the basering.

Example:

```
LIB "monomial.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.4.11.2 minbaseMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.3 gcdMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.4 lcmMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.5 membershipMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.6 intersectMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.7 quotientMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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]=wy2zt
↦ _[5]=wxyzt
↦ _[6]=w3xy
↦ _[7]=w3y2z
↦ _[8]=x2z4t3
↦ _[9]=wxz4t3
↦ _[10]=w2y2z2
```

D.4.11.8 radicalMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.9 isprimeMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.10 isprimaryMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.11 isirreducibleMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.12 isartinianMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.13 isgenericMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.14 dimMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 "monomial.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.4.11.15 irreddecMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 procesure 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 algorihtm of Roune (alg=lr),
- using the algorithm of Gao-Zhu (alg=gz).
- using the slice algorithm of Roune (alg=sr).

Example:

```
LIB "monomial.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.4.11.16 primdecMon

Procedure from library `monomial.lib` (see Section D.4.11 [monomial_lib], page 713).

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 procesure 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 algorihtm 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 "monomial.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]=wyz
↦    _[2]=y3
↦    _[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.4.12 mprimdec_lib

Library: mprimdec.lib

Purpose: procedures for primary decomposition of modules

Authors: Alexander Dreyer, dreyer@mathematik.uni-kl.de; adreyer@web.de

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.4.12.1 separator

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.2 PrimdecA

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.3 PrimdecB

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.4 modDec

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.5 zeroMod

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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(1)
↦       _[2]=gen(3)
↦       _[3]=z*gen(2)-gen(2)
↦    [2]:
↦       _[1]=z-1
↦ [2]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=gen(3)
↦       _[3]=z*gen(1)
↦    [2]:
↦       _[1]=z
↦ [3]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(2)
↦       _[3]=z*gen(3)+gen(3)
↦    [2]:
↦       _[1]=z+1
```

D.4.12.6 GTZmod

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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]=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.4.12.7 dec1var

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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(1)
↦       _[2]=gen(3)
↦       _[3]=z*gen(2)-gen(2)
↦    [2]:
↦       _[1]=z-1
↦ [2]:
↦    [1]:
↦       _[1]=gen(2)
↦       _[2]=gen(3)
↦       _[3]=z*gen(1)
↦    [2]:
↦       _[1]=z
↦ [3]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(2)
↦       _[3]=z*gen(3)+gen(3)
↦    [2]:
↦       _[1]=z+1
```

D.4.12.8 annil

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.9 splitting

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.10 primTest

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.11 preComp

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.12 indSet

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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.4.12.13 GTZopt

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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]=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.4.12.14 zeroOpt

Procedure from library `mprimdec.lib` (see Section D.4.12 [mprimdec_lib], page 724).

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]=gen(1)
↦       _[2]=z*gen(2)-gen(2)
↦       _[3]=gen(3)
↦    [2]:
↦       _[1]=z-1
↦ [2]:
↦    [1]:
↦       _[1]=z*gen(1)
↦       _[2]=gen(2)
↦       _[3]=gen(3)
↦    [2]:
↦       _[1]=z
↦ [3]:
↦    [1]:
↦       _[1]=gen(1)
↦       _[2]=gen(2)
↦       _[3]=z*gen(3)+gen(3)
↦    [2]:
↦       _[1]=z+1
```

D.4.13 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.4.13.1 regIdeal

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.13.2 depthIdeal

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.13.3 satiety

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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))[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.4.13.4 regMonCurve

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.13.5 NoetherPosition

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.13.6 is_NP

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.13.7 is_nested

Procedure from library `mregular.lib` (see Section D.4.13 [mregular_lib], page 733).

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.4.14 noether_lib

Library: noether.lib

Purpose: Noether normalization of an ideal (not nessecary 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.4.14.1 NPos_test

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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.

D.4.14.2 modNpos_test

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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).

D.4.14.3 NPos

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

Usage: NPos(i); i ideal

Return: A linear map phi such that phi(i) is in Noether position

D.4.14.4 modNPos

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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.

D.4.14.5 nsatiety

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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))[1] coincide.

D.4.14.6 modsatiety

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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))[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).

D.4.14.7 regCM

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

D.4.14.8 modregCM

Procedure from library `noether.lib` (see Section D.4.14 [noether_lib], page 738).

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).

D.4.15 normal_lib

Library: normal.lib

Purpose: Normalization of Affine Rings

Authors: G.-M. Greuel, greuel@mathematik.uni-kl.de,
S. Laplagne, slaplagn@dm.uba.ar,
G. Pfister, pfister@mathematik.uni-kl.de

Main procedures: Auxiliary procedures:

D.4.15.1 normal

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

Usage: normal(id [,choose]); id = radical ideal, choose = list of optional strings.
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 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.
If the optional parameter choose is not given or empty, only "equidim" but no other option is used.
The following options can be used when the ring has two variables. They are needed for computing integral basis.
- "var1" -> uses a polynomial in the first variable as conductor.

- "var2" -> uses a polynomial in the second variable as conductor.

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: a list, say nor, of size 2 (resp. 3 with option "withDelta"). @format 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).
@end format

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]:
↦       //   characteristic : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering dp
```

```
↦ //                   : names    x y
↦ //         block   3 : ordering C
↦ //    [2]:
↦ //       //   characteristic : 0
↦ //   number of vars : 3
↦ //         block   1 : ordering dp
↦ //                   : names    T(1)
↦ //         block   2 : ordering dp
↦ //                   : names    x y
↦ //         block   3 : ordering C
↦ //    [3]:
↦ //       //   characteristic : 0
↦ //   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]:
↦       //   characteristic : 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.4.15.3 [normalC], page 748; Section D.4.15.2 [normalP], page 746.

D.4.15.2 normalP

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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;
↦ // ** redefining gnirlist **
↦
↦ // '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]:
↦       //   characteristic : 2
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering dp
↦ //                  : names    y
↦ //        block   3 : ordering C
↦    [2]:
↦       //   characteristic : 2
↦ //   number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    x
↦ //        block   2 : ordering C
↦ [2]:
↦    [1]:
↦       _[1]=y3+xy
↦       _[2]=x2
↦    [2]:
↦       _[1]=1
↦ [3]:
↦    [1]:
↦       3,0
↦    [2]:
↦       6
def R2  = nor[1][2]; setring R2;
norid; normap;
↦ norid[1]=0
↦ normap[1]=x
↦ normap[2]=0
```

See also: Section D.4.15.1 [normal], page 740; Section D.4.15.3 [normalC], page 748.

D.4.15.3 normalC

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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]:
↦       //   characteristic : 0
↦ //   number of vars : 1
↦ //        block   1 : ordering dp
↦ //                  : names    T(1)
↦ //        block   2 : ordering C
↦    [2]:
↦       //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering a
↦ //                  : names    T(1) T(2)
↦ //                  : weights     1    0
↦ //        block   2 : ordering dp
↦ //                  : names    T(1) T(2)
↦ //        block   3 : ordering C
↦    [3]:
↦       //   characteristic : 0
↦ //   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)
printlevel = printlevel-1;
ring r = 2,(x,y,z),dp;
ideal i = z3-xy4;
nor = normalC(i);  nor;
↦
```

```
↦ // '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]:
↦       //   characteristic : 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
↦
↦ // '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]:
↦       //   characteristic : 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.4.15.1 [normal], page 740; Section D.4.15.2 [normalP], page 746.

D.4.15.4 HomJJ

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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;
↦ //   characteristic : 0
↦ //   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.4.15.5 genus

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

Usage: genus(i) or genus(i,1); I a 1-dimensional ideal

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: 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,1) uses the normalization to compute delta. Usually genus(i,1) is slower than genus(i) but sometimes not.

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);
↦ 5
```

D.4.15.6 primeClosure

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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);
↦ //   characteristic : 0
↦ //   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);
↦ //   characteristic : 0
↦ //   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(1)*X(6)-X(2)*X(7)
↦ ker[2]=X(2)*X(7)-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(7)*T(3)+X(6)*T(4)
↦ ker[14]=X(2)*X(4)*T(2)-X(6)*T(3)
↦ ker[15]=X(1)*X(4)-X(2)*T(3)
↦ ker[16]=X(1)*X(4)*T(2)-X(6)*T(4)
↦ ker[17]=X(2)*X(3)-X(1)*T(2)
↦ ker[18]=X(2)*X(3)*T(3)-X(6)*T(2)
↦ ker[19]=X(2)*X(3)*T(4)-X(7)*T(2)
↦ ker[20]=T(2)^2-X(3)*T(4)
↦ ker[21]=-X(3)*X(4)+T(2)*T(3)
↦ ker[22]=T(3)^2-X(4)*T(4)
↦ ker[23]=-X(3)*T(3)+T(2)*T(4)
↦ ker[24]=-X(4)*T(2)+T(3)*T(4)
↦ ker[25]=-X(3)*X(4)+T(4)^2
```

D.4.15.7 closureFrac

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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;
↦ // ** 'ker' is already global
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;
```

```
↦ // ** `f` is already global
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.4.15.8 iMult

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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.4.15.9 deltaLoc

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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.5.9.12 [delta], page 947; Section D.5.13.15 [tjurina], page 969.

D.4.15.10 locAtZero

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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.4.15.11 norTest

Procedure from library `normal.lib` (see Section D.4.15 [normal_lib], page 740).

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.4.16 normaliz_lib

Library: normaliz.lib

Purpose: Provides an interface for the use of Normaliz 2.2 within SINGULAR.

Authors: Winfried Bruns, Winfried.Bruns@Uni-Osnabrueck.de
Christof Soeger, Christof.Soeger@Uni-Osnabrueck.de

Overview: The library normaliz.lib provides an interface for the use of Normaliz 2.2 within SINGULAR. The exchange of data is via files, the only possibility offered by Normaliz in its present version. 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. Therefore SINGULAR can be used as a comfortable environment for the work with Normaliz.
Please see the `Normaliz2.2Documentation.pdf` and `nmz_sing.pdf` (both are included in the Normaliz distribution) for a more extensive documentation of Normaliz.

Singular and Normaliz exchange data via files. 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.

Use of this library requires the program Normaliz to be installed. You can download it from http://www.mathematik.uni-osnabrueck.de/normaliz/. Please make sure that the executables are in the search path or use setNmzExecPath (Section D.4.16.14 [setNmzExecPath], page 769).

Note: These library functions use `sed` to transfer the Normaliz output into a SINGULAR compliant format.

Procedures:

D.4.16.1 intclToricRing

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: intclToricRing(ideal I);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I. 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 one of the options `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

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
```

See also: Section D.4.16.3 [ehrhartRing], page 762; Section D.4.16.4 [intclMonIdeal], page 762; Section D.4.16.2 [normalToricRing], page 761.

D.4.16.2 normalToricRing

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: normalToricRing(ideal I);

Return: The toric ring S is the subalgebra of the basering generated by the leading monomials of the elements of I. 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 `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

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]=x3
↦ _[2]=x2y
↦ _[3]=xy2
↦ _[4]=y3
```

See also: Section D.4.16.3 [ehrhartRing], page 762; Section D.4.16.4 [intclMonIdeal], page 762; Section D.4.16.1 [intclToricRing], page 761.

D.4.16.3 ehrhartRing

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: ehrhartRing(ideal I);

Return: The exponent vectors of the leading monomials of the elements of I are considered as vertices 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 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 one of the options `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

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]=xy2t7
↦    _[2]=xy2t6
↦    _[3]=xy2t5
↦    _[4]=xy2t4
↦    _[5]=x2yt3
↦    _[6]=xy2t3
↦    _[7]=x2yt2
↦    _[8]=xy2t2
↦    _[9]=x2yt
↦    _[10]=xy2t
↦    _[11]=x3
↦    _[12]=x2y
↦    _[13]=xy2
↦    _[14]=y3
```

See also: Section D.4.16.4 [intclMonIdeal], page 762; Section D.4.16.1 [intclToricRing], page 761; Section D.4.16.2 [normalToricRing], page 761.

D.4.16.4 intclMonIdeal

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: intclMonIdeal(ideal I);

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 Definiton 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 one of the options `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

Note: A mathematical remark: the Rees algebra depends on the list of monomials given, and not only on the ideal they generate!

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]=x2
↦ _[2]=xy
↦ _[3]=y2
↦ _[4]=z3
↦ _[5]=xz2
↦ _[6]=yz2
l[2];  // monomials generating the integral closure of the Rees algebra
↦ _[1]=x2t
↦ _[2]=z
↦ _[3]=xyt
↦ _[4]=y2t
↦ _[5]=x
↦ _[6]=z3t
↦ _[7]=y
↦ _[8]=xz2t
↦ _[9]=yz2t
```

See also: Section D.4.16.3 [ehrhartRing], page 762; Section D.4.16.1 [intclToricRing], page 761; Section D.4.16.2 [normalToricRing], page 761.

D.4.16.5 torusInvariants

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: torusInvariants(intmat A);

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 T if one of the options `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

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 $A = (a_{i,j})$.

Note: It is of course possible that $R^T = K$. At present, Normaliz cannot deal with the zero cone and will issue the (wrong) error message that the cone is not pointed. The function also gives an error message if the matrix T has the wrong number of columns.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,w),dp;
intmat V0[2][4]=0,1,2,3, -1,1,2,1;
valRing(V0);
↦ _[1]=y
↦ _[2]=xy
↦ _[3]=w
↦ _[4]=xw
↦ _[5]=z
↦ _[6]=xz
↦ _[7]=x2z
```

See also: Section D.4.16.6 [valRing], page 764; Section D.4.16.7 [valRingIdeal], page 765.

D.4.16.6 valRing

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: valRing(intmat V);

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`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

Note: It is of course possible that $S = K$. At present, Normaliz cannot deal with the zero cone and will issue the (wrong) error message that the cone is not pointed. 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 V0[2][4]=0,1,2,3, -1,1,2,1;
valRing(V0);
↦ _[1]=y
↦ _[2]=xy
↦ _[3]=w
↦ _[4]=xw
↦ _[5]=z
↦ _[6]=xz
↦ _[7]=x2z
```

See also: Section D.4.16.5 [torusInvariants], page 763; Section D.4.16.7 [valRingIdeal], page 765.

D.4.16.7 valRingIdeal

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: valRingIdeal(intmat V);

Return: The function returns two ideals, both to be considered as lists of monomials which generate an algebra over the coefficient field. 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 input matrix T if one of the options `supp`, `triang`, or `hvect` 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.4.16.8 [showNuminvs], page 766, Section D.4.16.9 [exportNuminvs], page 766).

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: It is of course possible that $S = K$. At present, Normaliz cannot deal with the zero cone and will issue the (wrong) error message that the cone is not pointed. The function also gives an error message if the matrix T 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;
valRingIdeal(V);
↦ [1]:
↦    _[1]=y
↦    _[2]=xy
```

```
↦     _[3]=w
↦     _[4]=xw
↦     _[5]=z
↦     _[6]=xz
↦     _[7]=x2z
↦  [2]:
↦     _[1]=zw
↦     _[2]=xz2
↦     _[3]=z2
↦     _[4]=y2w
↦     _[5]=y2z
↦     _[6]=xy2z
↦     _[7]=y4
↦     _[8]=xy4
↦     _[9]=yw2
↦     _[10]=w3
```

See also: Section D.4.16.5 [torusInvariants], page 763; Section D.4.16.6 [valRing], page 764.

D.4.16.8 showNuminvs

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: showNuminvs();

Purpose: prints the numerical invariants

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,t),dp;
ideal I=x^2,y^2,z^3;
list l=intclMonIdeal(I);
showNuminvs();
↦ hilbert_basis_elements : 9
↦ number_extreme_rays : 6
↦ rank : 4
↦ index : 1
↦ number_support_hyperplanes : 5
↦ homogeneous : 0
↦ primary : 1
↦ ideal_multiplicity : 12
```

See also: Section D.4.16.9 [exportNuminvs], page 766.

D.4.16.9 exportNuminvs

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.8 [showNuminvs], page 766) to see which invariants are available.

Example:

```
LIB "normaliz.lib";
ring R=0,(x,y,z,t),dp;
```

```
ideal I=x^2,y^2,z^3;
list l=intclMonIdeal(I);
exportNuminvs();
// now the following variables are set:
nmz_hilbert_basis_elements;
↦ 9
nmz_number_extreme_rays;
↦ 6
nmz_rank;
↦ 4
nmz_index;
↦ 1
nmz_number_support_hyperplanes;
↦ 5
nmz_homogeneous;
↦ 0
nmz_primary;
↦ 1
nmz_ideal_multiplicity;
↦ 12
```

See also: Section D.4.16.8 [showNuminvs], page 766.

D.4.16.10 setNmzOption

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: setNmzOption(string s, int onoff);

Purpose: If `onoff=1` the option `s` is activated, and if `onoff=0` it is deactivated. The Normaliz options are accessible via the following names:

```
-s: supp
-v: triang
-p: hvect
-n: normal
-h: hilb
-d: dual
-a: allf
-c: control
-i: ignore
-e: errorcheck
```

Example:

```
LIB "normaliz.lib";
setNmzOption("hilb",1);
↦ 1
showNmzOptions();
↦  -f -h -i
```

See also: Section D.4.16.11 [showNmzOptions], page 767.

D.4.16.11 showNmzOptions

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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 -i
```

See also: Section D.4.16.10 [setNmzOption], page 767.

D.4.16.12 normaliz

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: normaliz(intmat sgr,int nmz_mode);

Return: The function applies Normaliz to the parameter sgr in the mode set by nmz_mode. The function returns the intmat defined by the file with suffix gen.

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,1);
↦ 1,1,
↦ 2,1,
↦ 1,2,
↦ 3,1,
↦ 1,3
```

See also: Section D.4.16.3 [ehrhartRing], page 762; Section D.4.16.4 [intclMonIdeal], page 762; Section D.4.16.1 [intclToricRing], page 761; Section D.4.16.2 [normalToricRing], page 761; Section D.4.16.5 [torusInvariants], page 763; Section D.4.16.6 [valRing], page 764; Section D.4.16.7 [valRingIdeal], page 765.

D.4.16.13 setNmzVersion

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: setNmzVersion(string s); `s` version of the Normaliz executable

Create: `Normaliz::nmz_version` to save the given version `s`

Note: The version coincides with the filename of the Normaliz executable. Possible arguments are:
`norm32` for 32bit integer precision
`norm64` for 64bit integer precision (default)
`normbig` for arbitrary precision

Example:

```
LIB "normaliz.lib";
setNmzVersion("normbig");
```

See also: Section D.4.16.14 [setNmzExecPath], page 769.

D.4.16.14 setNmzExecPath

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.13 [setNmzVersion], page 768.

D.4.16.15 writeNmzData

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

Usage: writeNmzData(intmat M, int mode);

Create: Creates an input file for Normaliz from the matrix M. The second parameter sets the mode. How the matrix is interpreted depends on the mode. See the Normaliz documentation for more information.

Note: Needs an explicit filename set. The filename is created from the current filename and the suffix given to the function.
Note that all 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,1);
int dummy=system("sh","cat VeryInteresting.in");
↦ 3
↦ 3
↦ 1 2 3
↦ 4 5 6
↦ 7 8 10
↦ 1
```

See also: Section D.4.16.16 [readNmzData], page 769; Section D.4.16.21 [rmNmzFiles], page 772; Section D.4.16.18 [setNmzDataPath], page 771; Section D.4.16.17 [setNmzFilename], page 770.

D.4.16.16 readNmzData

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.

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 uses the command `sed` to transfer the normaliz output into a singular conform format.

Example:

```
LIB "normaliz.lib";
setNmzFilename("VeryInteresting");
intmat sgr[3][3]=1,2,3,4,5,6,7,8,10;
intmat sgrnormal=normaliz(sgr,0);
readNmzData("sup");
↦ -2,-2,3,
↦ -4,11,-6,
↦ 1,-2,1
readNmzData("typ");
↦ 3,0,0,
↦ 2,1,0,
↦ 1,2,0,
↦ 0,3,0,
↦ 0,0,1
```

See also: Section D.4.16.21 [rmNmzFiles], page 772; Section D.4.16.18 [setNmzDataPath], page 771; Section D.4.16.17 [setNmzFilename], page 770; Section D.4.16.15 [writeNmzData], page 769.

D.4.16.17 setNmzFilename

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.15 [writeNmzData], page 769 and Section D.4.16.16 [readNmzData], page 769).
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.4.16.16 [readNmzData], page 769; Section D.4.16.21 [rmNmzFiles], page 772; Section D.4.16.18 [setNmzDataPath], page 771; Section D.4.16.15 [writeNmzData], page 769.

D.4.16.18 setNmzDataPath

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.16 [readNmzData], page 769; Section D.4.16.21 [rmNmzFiles], page 772; Section D.4.16.17 [setNmzFilename], page 770; Section D.4.16.15 [writeNmzData], page 769.

D.4.16.19 writeNmzPaths

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.18 [setNmzDataPath], page 771; Section D.4.16.14 [setNmzExecPath], page 769; Section D.4.16.20 [startNmz], page 771.

D.4.16.20 startNmz

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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";
startNmz();
↦ nmz_exec_path is ../Normaliz/
↦ nmz_data_path not set
```

See also: Section D.4.16.18 [setNmzDataPath], page 771; Section D.4.16.14 [setNmzExecPath], page 769; Section D.4.16.19 [writeNmzPaths], page 771.

D.4.16.21 rmNmzFiles

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.17 [setNmzFilename], page 770).

Example:

```
LIB "normaliz.lib";
setNmzFilename("VeryInteresting");
rmNmzFiles();
```

See also: Section D.4.16.16 [readNmzData], page 769; Section D.4.16.18 [setNmzDataPath], page 771; Section D.4.16.17 [setNmzFilename], page 770; Section D.4.16.15 [writeNmzData], page 769.

D.4.16.22 mons2intmat

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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.4.16.23 [intmat2mons], page 772.

D.4.16.23 intmat2mons

Procedure from library `normaliz.lib` (see Section D.4.16 [normaliz_lib], page 760).

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 5.1.25 [ERROR], page 145).

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.4.16.22 [mons2intmat], page 772.

D.4.17 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.4.17.1 nonMonomials

Procedure from library `pointid.lib` (see Section D.4.17 [pointid_lib], page 773).

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 rp, i.e., be of the form 0,x(1..n),rp; (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),rp;
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),rp;
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.4.17.2 cornerMonomials

Procedure from library `pointid.lib` (see Section D.4.17 [pointid_lib], page 773).

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),rp;
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.4.17.3 facGBIdeal

Procedure from library `pointid.lib` (see Section D.4.17 [pointid_lib], page 773).

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 rp, i.e., be of the form 0,x(1..n),rp; (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),rp;
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.4.18 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 ideas 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.

Procedures:

D.4.18.1 Ann

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.2 primdecGTZ

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

Usage: primdecGTZ(i); i ideal

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.

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!)

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]=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-z2
```

D.4.18.3 primdecSY

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

Usage: primdecSY(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.

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.

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.4.18.4 minAssGTZ

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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 I.

Note: Designed for characteristic 0, works also in char k > 0 based on an algorithm of Yokoyama

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]=z2+1
↦    _[2]=-z2+y
↦ [2]:
↦    _[1]=z3+2
↦    _[2]=-z2+y
```

D.4.18.5 minAssChar

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

Usage: minAssChar(I[,c]); i ideal, c int (optional).

Return: list, the minimal associated prime ideals of i.

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.

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.4.18.6 testPrimary

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.7 radical

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.8 radicalEHV

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.9 equiRadical

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.10 prepareAss

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

Usage: prepareAss(I); I ideal

Return: list, the radicals of the maximal dimensional 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.4.18.11 equidim

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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
- a[1],...,a[s-1] the lower dimensional equidimensional loci.

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.4.18.12 equidimMax

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.13 equidimMaxEHV

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.14 zerodec

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.4.18.15 absPrimdecGTZ

Procedure from library `primdec.lib` (see Section D.4.18 [primdec_lib], page 779).

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.

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);
↦
↦ // def S = absPrimdecGTZ(i); creates a ring,
↦ // which comes with two lists:
↦ // absolute_primes -- the absolute prime components,
↦ // and primary_decomp -- the primary and prime
↦ // components over the current basering).
↦ // Type setring S; absolute_primes;
↦ // to access the data.
↦
setring S;
absolute_primes;
↦ [1]:
↦    [1]:
↦       _[1]=a3+2
↦       _[2]=z-a
↦       _[3]=y-za
↦    [2]:
↦       3
↦ [2]:
↦    [1]:
↦       _[1]=a2+1
↦       _[2]=z-a
↦       _[3]=y+1
↦    [2]:
↦       2
```

See also: Section D.4.1.1 [absFactorize], page 652; Section D.4.18.2 [primdecGTZ], page 780.

D.4.19 primitiv_lib

Library: primitiv.lib

Purpose: Computing a Primitive Element

Author: Martin Lamm, email: lamm@mathematik.uni-kl.de

Procedures:

D.4.19.1 primitive

Procedure from library `primitiv.lib` (see Section D.4.19 [primitiv_lib], page 787).

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.4.19.2 [primitive_extra], page 788.

D.4.19.2 primitive_extra

Procedure from library `primitiv.lib` (see Section D.4.19 [primitiv_lib], page 787).

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+(-y5-y4-y3-y2-y-1)
↦    _[3]=x+(y3+y+1)
↦    _[4]=x+(y5+y4+y2+1)
↦ [2]:
↦    1,1,1,1
```

D.4.19.3 splitring

Procedure from library `primitiv.lib` (see Section D.4.19 [primitiv_lib], page 787).

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))
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : (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;
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : (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.4.20 realrad_lib

Library: realrad.lib

Purpose: Computation of real radicals

Author : Silke Spang

Overview: Algorithms about the computation of the real
radical of an arbitary ideal over the rational numbers and transcendetal extensions thereof

Procedures:

D.4.20.1 realpoly

Procedure from library `realrad.lib` (see Section D.4.20 [realrad_lib], page 790).

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.4.20.2 realzero

Procedure from library `realrad.lib` (see Section D.4.20 [realrad_lib], page 790).

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.4.20.3 realrad

Procedure from library `realrad.lib` (see Section D.4.20 [realrad_lib], page 790).

Usage: realrad(id), id an ideal of arbitary 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]=x4-x2y2+x2y-y3
↦ _[2]=y5+y4z+5y4+5y3z+7y3+7y2z+3y2+3yz+2y+2z
↦ _[3]=xy4+5xy3+7xy2+3xy+2x+y4z+5y3z+7y2z+3yz+2z
↦ _[4]=y4z3+2y4+5y3z3+10y3+7y2z3+14y2+3yz3+6y+2z3+4
```

D.4.21 reesclos_lib

Library: reesclos.lib

Purpose: procedures to compute the int. closure of an ideal

Author: Tobias Hirsch, email: hirsch@math.tu-cottbus.de

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] (in the same manner as done in the library 'normal.lib'), which is a graded subalgebra of R[t]. The degree-k-component is the integral closure of the k-th power of I.

Procedures:

D.4.21.1 ReesAlgebra

Procedure from library `reesclos.lib` (see Section D.4.21 [reesclos_lib], page 792).

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;
↦ //   characteristic : 0
↦ //   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.4.21.2 normalI

Procedure from library `reesclos.lib` (see Section D.4.21 [reesclos_lib], page 792).

Usage: normalI (I [,p[,r]]); I an ideal, p and r 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.
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]=y5
↦     _[3]=-xy3
```

D.4.22 resbin_lib

Library: resbin.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: See also: Section D.4.23 [resolve_lib], page 843.

D.4.22.1 BINresol

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [2]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [3]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2)
↦ //        block   2 : ordering C
↦ [4]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2)
↦ //        block   2 : ordering C
```

```
B[2]; // list of all charts
↦ [1]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [2]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [3]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [4]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2)
↦ //        block   2 : ordering C
↦ [5]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2)
↦ //        block   2 : ordering C
↦ [6]:
↦    //   characteristic : 0
↦ //   number of vars : 2
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2)
↦ //        block   2 : ordering C
↦ [7]:
↦    //   characteristic : 0
↦ //   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 **
ideal J=x(1)^2-x(2)^2*x(3)^2;
list B=BINresol(J);
↦ // ** redefining B **
B[2]; // list of all charts
↦ [1]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
```

```
↦ //                 : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [2]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [3]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [4]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [5]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [6]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [7]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) y(3)
↦ //        block   2 : ordering C
↦ [8]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [9]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) y(2) x(3)
↦ //        block   2 : ordering C
↦ [10]:
↦    //   characteristic : 2
↦ //   number of vars : 3
```

```
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2) x(3)
↦ //        block   2 : ordering C
↦ [11]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    x(1) x(2) y(3)
↦ //        block   2 : ordering C
↦ [12]:
↦    //   characteristic : 2
↦ //   number of vars : 3
↦ //        block   1 : ordering dp
↦ //                  : names    y(1) x(2) x(3)
↦ //        block   2 : ordering C
```

D.4.22.2 Eresol

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: Eresol(J); J ideal

Return: The E-resolution of singularities of J in terms of the affine charts, see example

Example:

```
LIB "resbin.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.4.22.3 determinecenter

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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,list0(4),list0(4),list0(4),iniD(4),iniD(4),list0(4),-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 acumulated
↦ [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 **
list flag=identifyvar();
↦ // ** redefining flag **
ideal J=x(1)^3-x(2)^2*x(3)^5, x(1)*x(3)^3+x(4)^5;
list Lmb=2,list0(4),list0(4),list0(4),iniD(4),iniD(4),list0(4),-1;
↦ // ** redefining Lmb **
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.4.22.4 Blowupcenter

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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,list0(7),list0(7),list0(7),iniD(7),iniD(7),list0(7),-1;
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,list0(7),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.4.22.5 Nonhyp

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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 **
list flag=identifyvar();
↦ // ** redefining flag **
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.4.22.6 inidata

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: inidata(K,k); K any ideal, k integer (!=0)

Compute: Verifies the input data

Return: flag indicating if the ideal is binomial or not

Example:

```
LIB "resbin.lib";
ring r = 0,(x(1..3)),dp;
ideal J1=x(1)^4*x(2)^2, x(1)^2+x(3)^3;
inidata(J1,2);
↦ 1
ideal J2=x(1)^4*x(2)^2, x(1)^2+x(2)^3+x(3)^5;
inidata(J2,2);
↦ 0
```

D.4.22.7 identifyvar

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: identifyvar();

Compute: Asign 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 "resbin.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.4.22.8 data

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: data(K,k,n); K any ideal, k integer (!=0), n integer (!=0)

Compute: Construcs a list with the coefficients and exponents of one ideal

Return: lists of coefficients and exponents of K

Example:

```
LIB "resbin.lib";
ring r = 0,(x(1..3)),dp;
ideal J=x(1)^4*x(2)^2, x(1)^2-x(3)^3;
data(J,2,3);
↦ [1]:
↦    [1]:
↦       1
↦ [2]:
↦    [1]:
↦       -1
↦    [2]:
↦       1
↦ [1]:
↦    [1]:
↦       [1]:
↦          4
↦       [2]:
↦          2
↦       [3]:
```

```
↦           0
↦ [2]:
↦    [1]:
↦       [1]:
↦          0
↦       [2]:
↦          0
↦       [3]:
↦          3
↦    [2]:
↦       [1]:
↦          2
↦       [2]:
↦          0
↦       [3]:
↦          0
```

D.4.22.9 Edatalist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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 **
list flag=identifyvar();
```

```
↦ // ** redefining flag **
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 **
list flag=identifyvar();
↦ // ** redefining flag **
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.4.22.10 EOrdlist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.11 maxEord

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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(
list L=data(J,4,8);
list M=maxEord(L[1],L[2],4,8,flag);
M[1];    // E-order
↦ 1
```

D.4.22.12 ECoef

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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 **
list flag=identifyvar();
↦ // ** redefining flag **
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 **
list flag=identifyvar();
↦ // ** redefining flag **
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 **
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 **
list flag=identifyvar();
↦ // ** redefining flag **
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.4.22.13 elimrep

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: elimrep(L); L is a list

Compute: Eliminate repeated terms from a list

Return: the same list without repeated terms

Example:

```
LIB "resbin.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.4.22.14 Emaxcont

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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*
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 **
list flag=identifyvar();
↦ // ** redefining flag **
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.4.22.15 cleanunit

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.16 resfunction

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: resfunction(invariant,auxinv,nchart,n);
invariant, auxinv lists, nchart, n integers

Compute: Patch the resolution function

Return: The complete resolution function

Example:

```
LIB "resbin.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.4.22.17 calculateI

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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,list0(3),list0(3),list0(3),iniD(3),iniD(3),list0(3),-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,list0(3),auxpath,list(),list();
list L3=Blowupcenter(LL[1],1,1,infochart,3,3,0); // blowing-up and looking to the x(3
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.4.22.18 Maxord

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.19 Gamma

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.20 convertdata

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.21 tradblwup

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

D.4.22.22 lcmofall

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.23 computemcm

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.24 constructH

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: constructH(Hhist,n,flag);
Hhist intvec, n integer, flag list

Return: the list of exceptional divisors accumulated at this chart

Example:

```
LIB "resbin.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.4.22.25 constructblwup

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.26 constructlastblwup

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.27 genoutput

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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: command not found
↦ sh: /usr/bin/X11/xv: No such file or director
↦ Press <Return> to continue
↦ 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.4.22.28 salida

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.29 iniD

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: iniD(n); n integer

Return: list of lists of zeros of size n

Example:

```
LIB "resbin.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.4.22.30 sumlist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: sumlist(L1,L2); L1,L2 lists, (size(L1)==size(L2))

Return: a list, sum of L1 and L2

Example:

```
LIB "resbin.lib";
list L1=1,2,3;
list L2=5,9,7;
sumlist(L1,L2);
↦ [1]:
↦    6
↦ [2]:
↦    11
↦ [3]:
↦    10
```

D.4.22.31 reslist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: reslist(L1,L2); L1,L2 lists, (size(L1)==size(L2))

Return: a list, subtraction of L1 and L2

Example:

```
LIB "resbin.lib";
list L1=1,2,3;
list L2=5,9,7;
reslist(L1,L2);
↦ [1]:
↦    -4
↦ [2]:
↦    -7
↦ [3]:
↦    -4
```

D.4.22.32 multiplylist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: multiplylist(L,a); L list, a number

Return: list of elements of type number, multiplication of L times a

Example:

```
LIB "resbin.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.4.22.33 dividelist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: dividelist(L1,L2); L1,L2 lists

Return: list of elements of type number, division of L1 by L2

Example:

```
LIB "resbin.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.4.22.34 createlist

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

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 "resbin.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.4.22.35 list0

Procedure from library `resbin.lib` (see Section D.4.22 [resbin_lib], page 794).

Usage: list0(n); n integer

Return: list of n zeros

Example:

```
LIB "resbin.lib";
list0(4);
↦ [1]:
↦    0
↦ [2]:
↦    0
↦ [3]:
↦    0
↦ [4]:
↦    0
```

D.4.23 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

Main procedures: Procedures for pretty printing of output: Auxillary procedures:

D.4.23.1 blowUp

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.2 blowUp2

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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;
↦ //   characteristic : 0
↦ //   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;
↦ //   characteristic : 0
↦ //   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.4.23.3 Center

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.4 resolve

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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 throught 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 occured 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.4.23.5 showBO

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.6 presentTree

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

Usage: presentTree(L);
L=list, output of resolve

Return: nothing, only pretty printing of the output data of resolve()

D.4.23.7 showDataTypes

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

Usage: showDataTypes();

Return: nothing, only pretty printing of extended version of help text

D.4.23.8 blowUpBO

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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 exeptional 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.4.23.9 createBO

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.10 CenterBO

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.11 Delta

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.23.12 DeltaList

Procedure from library `resolve.lib` (see Section D.4.23 [resolve_lib], page 843).

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.4.24 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

Main procedures: Auxilliary procedures:

D.4.24.1 intersectionDiv

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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]:
↦    -4,1,1,
↦    1,-3,1,
↦    1,1,-2
↦ [2]:
↦    0,0,0
↦ [3]:
↦    [1]:
↦       [1]:
↦          3,1,1
↦       [2]:
↦          4,1,1
↦       [3]:
↦          5,1,1
↦    [2]:
↦       [1]:
↦          3,1,2
↦       [2]:
↦          5,1,2
↦    [3]:
↦       [1]:
↦          4,2,1
↦       [2]:
↦          5,2,1
↦ [4]:
↦    1,1,1
```

D.4.24.2 spectralNeg

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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"gaussman.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.4.24.3 discrepancy

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

Usage: discrepancy(L);
L = list of rings

Assume: L is the output of resolution of singularities

Retrun: 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.4.24.4 zetaDL

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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 multiplicites as needed in computation of zeta function
l[4]: intvec N of multiplicities as needed in compuation 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.4.24.5 collectDiv

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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.4.24.6 prepEmbDiv

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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.4.24.7 abstractR

Procedure from library `reszeta.lib` (see Section D.4.24 [reszeta_lib], page 852).

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.4.25 sagbi_lib

Library: sagbi.lib

Purpose: Compute subalgebra bases analogous to Groebner bases for ideals

Authors: Gerhard Pfister, pfister@mathematik.uni-kl.de,
Anen Lakhal, alakhal@mathematik.uni-kl.de

Procedures:

D.4.25.1 sagbiRreduction

Procedure from library `sagbi.lib` (see Section D.4.25 [sagbi_lib], page 861).

D.4.25.2 sagbiSPoly

Procedure from library `sagbi.lib` (see Section D.4.25 [sagbi_lib], page 861).

Usage: sagbiSPoly(id [,n]); id ideal, n positive integer.

Return: an ideal

- If (n=0 or default) an ideal, whose generators are the S-polynomials.
- If (n=1) a list of size 2:
  the first element of this list is the ideal of S-polynomials.
  the second element of this list is the ring in which the ideal of algebraic relations is defined.

Example:

```
LIB "sagbi.lib";
ring r=0, (x,y),dp;
poly f1,f2,f3,f4=x2,y2,xy+y,2xy2;
ideal I=f1,f2,f3,f4;
sagbiSPoly(I);
↦ _[1]=xy2+1/2y2
↦ _[2]=xy4+1/2y4
↦ _[3]=x3y4+3/2x2y4+xy4+1/4y4
list L=sagbiSPoly(I,1);
↦
↦ // 'sagbiSPoly' created a ring as 2nd element of the list.
↦ // The ring contains the ideal 'kern'  of algebraic relations between the
↦ //leading terms of the generators of I.
↦ // To access to this ring and see 'kern' you should give the ring a name,
↦ // e.g.:
↦                 def S = L[2]; setring S; kern;
↦
L[1];
```

```
↦ _[1]=xy2+1/2y2
↦ _[2]=xy4+1/2y4
↦ _[3]=x3y4+3/2x2y4+xy4+1/4y4
def S= L[2]; setring S; kern;
↦ kern[1]=@y(1)*@y(2)-@y(3)^2
↦ kern[2]=4*@y(2)*@y(3)^2-@y(4)^2
↦ kern[3]=4*@y(3)^4-@y(1)*@y(4)^2
```

D.4.25.3 sagbiNF

Procedure from library `sagbi.lib` (see Section D.4.25 [sagbi_lib], page 861).

Usage: sagbiNF(id,dom,k[,n]); id either poly or ideal,dom ideal, k and n positive intergers.

Return: same as type of id; ideal or polynomial.

The integer k determines what kind of s-reduction is performed:
- if (k=0) no tail s-reduction is performed.
- if (k=1) tail s-reduction is performed.

Three Algorithm variants are used to perform subalgebra reduction.
The positive integer n determines which variant should be used.
n may take the values (0 or default),1 or 2.

Note: computation of subalgebra normal forms may be performed in polynomial rings or quotients thereof

Example:

```
LIB "sagbi.lib";
ring r=0,(x,y),dp;
ideal I= x2-xy;
qring Q=std(I);
ideal dom =x2,x2y+y,x3y2;
poly p=x4+x2y+y;
sagbiNF(p,dom,0);
↦ 0
sagbiNF(p,dom,1);// tail subalgebra reduction is performed
↦ 0
```

D.4.25.4 sagbi

Procedure from library `sagbi.lib` (see Section D.4.25 [sagbi_lib], page 861).

Usage: sagbi(id,k[,n]); id ideal, k and n positive integers.

Return: A SAGBI basis for the subalgebra defined by the generators of id.

k determines what kind of s-reduction is performed:
- if (k=0) no tail s-reduction is performed.
- if (k=1) tail s-reduction is performed, and S-interreduced SAGBI basis is returned.

Three algorithm variants are used to perform subalgebra reduction.
The positive interger n determine which variant should be used.
n may take the values (0 or default),1 or 2.

Note: SAGBI bases computations may be performed in polynomial rings or quotients thereof.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y),dp;
ideal I=x2,y2,xy+y;
sagbi(I,1,1);
↦ _[1]=x2
↦ _[2]=y2
↦ _[3]=xy+y
↦ _[4]=xy2
```

D.4.25.5 sagbiPart

Procedure from library `sagbi.lib` (see Section D.4.25 [sagbi_lib], page 861).

Usage: sagbiPart(id,k,c[,n]); id ideal, k, c and n positive integers

Return: A partial SAGBI basis for the subalgebra defined by the generators of id.

k determines what kind of s-reduction is performed:
- if (k=0) no tail s-reduction is performed.
- if (k=1) tail s-reduction is performed, and S-intereduced SAGBI basis is returned.

c determines, after how many loops the Sagbi basis computation should stop.
Three algorithm variants are used to perform subalgebra reduction.
The positive integer n determines which variant should be used.
n may take the values (0 or default),1 or 2.

Note: - SAGBI bases computations may be performed in polynomial rings or quotients thereof.
- This version of sagbi is interesting in the case of subalgebras with infinte SAGBI basis. In this case, it may be used to check, if the elements of this basis have a particular form.

Example:

```
LIB "sagbi.lib";
ring r= 0,(x,y),dp;
ideal I=x,xy-y2,xy2;//the corresponding Subalgebra has an infinte SAGBI basis
sagbiPart(I,1,3);// computations should stop after 3 turns.
↦ _[1]=x
↦ _[2]=xy-y2
↦ _[3]=xy2
↦ _[4]=xy3-1/2y4
↦ _[5]=xy5-1/3y6
↦ _[6]=xy4
↦ _[7]=xy9-1/5y10
↦ _[8]=xy8
↦ _[9]=xy7-1/4y8
↦ _[10]=xy6
↦ _[11]=xy15-1/8y16
↦ _[12]=xy16
↦ _[13]=xy12
↦ _[14]=xy13-1/7y14
↦ _[15]=xy11-1/6y12
↦ _[16]=xy10
↦ _[17]=xy17-1/9y18
↦ _[18]=xy14
```

D.4.26 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: Auxiliary procedures:

D.4.26.1 truncate

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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,0, 0, x, 0, 0,
↦ 0,0,0,0,z,0,0, 0, 0,-z,y,0,0, -z,-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.4.26.2 truncateFast

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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 shoud 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, 0,0,0,0,0,0,0,0, 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,0, z,\
    0, 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,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, 0,0,0,0,0,0,0,0, 0, 0, 0, y, 0, 0, 0, 0, 0, 0, 0, 0,0,0,-v,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, 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, -v,0,\
    0, 0,
↦    -u,z, 0, 0, 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, 0, z, 0, 0,0,0,0,0,0,0,0, -v,0, 0, 0, -u,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, 0, z, 0,0,0,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, y, 0, 0, 0, 0,0,0,0, -v\
   ,0, 0,
↦    0, -u,0, 0, x, 0, 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,
↦ 0,0,0,0,0,0,0,0,0,0,0,v,0,0,0,0,0,0, 0, 0, 0, 0, 0,0,0,0,0,u,0,0,0,0,0,0, 0, 0,\
    -v,0,
↦    0, 0, -u,0, 0,z,0,0,0,0,0,0, 0, -v,0, 0, 0, -u,0, 0, -z,y, 0, 0,0,0,0, 0,\
    0, 0,
↦    0, 0, 0, 0, 0, 0, x, 0, 0, 0, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,v,0,0,0,0,0, 0, 0, 0, 0, 0,0,0,0,0,0,u,0,0,0,0,0, 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, 0, y, 0,0,0,0, 0,\
    -v,0,
↦    0, 0, -u,0, 0, -z,0, x, 0, 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, 0,0,0,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,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,v,0,0, 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, -v,
↦    0, 0, 0, -u,0, 0, -z,0, -y,x, 0,
↦ 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,v,0, 0, 0, 0, 0, 0,0,0,0,0,0,0,0,0,0,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, 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\
   , 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.4.26.3 CM_regularity

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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.4.26.4 sheafCohBGG

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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:

```
          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;
def A=sheafCohBGG(M,-9,4);
↦       -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);
def B=sheafCohBGG(M,-8,4);
↦        -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.4.26.7 [dimH], page 874; Section D.4.26.6 [sheafCoh], page 873.

D.4.26.5 sheafCohBGG2

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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:

```
            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;
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:
↦         -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:  54
//////////////////////////////////////////////
t = timer;
A = sheafCohBGG(M, l, h);  // BGG method (without optimization):
↦         -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:  101
//////////////////////////////////////////
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
"Time: ", timer - t;
↦ Time:  51
//////////////////////////////////////////
printlevel = pl;
kill A, r;
//------------------------------------------
// cohomology of cotangential bundle on P^3:
//------------------------------------------
ring R=32001,(x,y,z,u),dp;
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:
↦        -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:  8
//////////////////////////////////////////////
t = timer;
B = sheafCohBGG(M, l, h);  // BGG method (without optimization):
↦       -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:  9
//////////////////////////////////////////////
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
"Time: ", timer - t;
↦ Time:  3
//////////////////////////////////////////////
printlevel = pl;
```

See also: Section D.4.26.4 [sheafCohBGG], page 868.

D.4.26.6 sheafCoh

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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 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))
```

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;
def A=sheafCoh(0,-7,2);
↦       -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);
def B=sheafCoh(M,-6,2);
↦       -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.4.26.7 [dimH], page 874; Section D.4.26.4 [sheafCohBGG], page 868.

D.4.26.7 dimH

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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.4.26.6 [sheafCoh], page 873; Section D.4.26.4 [sheafCohBGG], page 868.

D.4.26.8 dimGradedPart

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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.4.26.9 displayCohom

Procedure from library `sheafcoh.lib` (see Section D.4.26 [sheafcoh_lib], page 864).

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.4.27 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

Requires: External programs 4ti2, sed and awk to be installed

Procedures:

D.4.27.1 markov4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.27 [sing4ti2_lib], page 875).

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);
↦ ------------------------------------------------
↦ 4ti2 version 1.3.2, Copyright (C) 2006 4ti2 team.
↦ 4ti2 comes with ABSOLUTELY NO WARRANTY.
↦ This is free software, and you are welcome
↦ to redistribute it under certain conditions.
↦ For details, see the file COPYING.
↦ ------------------------------------------------
↦ Using 64 bit integers.
↦ Computing generating set (Hybrid) ...
↦ Phase 1:
↦ Computing generating set (Saturation) ...
↦ Saturating 2 variable(s).
↦
  Sat   2: Col:   1 (U) Size:        1, ToDo:        1
  Sat   2: Col:  \
    1 (U) Size:        1, ToDo:        0
  Sat   2: Col:   1 (U) Size:      \
   1, Time:  0.00 /  0.00 secs.
↦   Saturated already on 1 variable(s).
↦ Done. Size:      1, Time: 0.00 /  0.00 secs
↦ Phase 2:
```

```
↦ Lifting 1 variable(s).
↦
  Lift   1: Col:   2 (F) Size:      1, Index:      0
  Lift   1: Col:   \
   2 (F) Size:      1, Time:  0.00 /  0.00 secs.
↦ Done. Size:      1, Time: 0.00 /  0.00 secs
↦ Computing Miminal Generation Set (Fast)...
↦
  Size:      1, Time:  0.00 /  0.00 secs. Done.
↦ 4ti2 Total Time:  0.00 secs.
↦ _[1]=-y2+xz
matrix N[1][3]=1,2,1;
markov4ti2(N,1);
↦ -----------------------------------------------
↦ 4ti2 version 1.3.2, Copyright (C) 2006 4ti2 team.
↦ 4ti2 comes with ABSOLUTELY NO WARRANTY.
↦ This is free software, and you are welcome
↦ to redistribute it under certain conditions.
↦ For details, see the file COPYING.
↦ -----------------------------------------------
↦ Using 64 bit integers.
↦ Computing generating set (Hybrid) ...
↦ 4ti2 Total Time:  0.00 secs.
↦ _[1]=xy2z-1
↦ _[2]=xy2z-1
```

D.4.27.2 hilbert4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.27 [sing4ti2_lib], page 875).

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);
↦ -----------------------------------------------
↦ 4ti2 version 1.3.2, Copyright (C) 2006 4ti2 team.
```

```
↦ 4ti2 comes with ABSOLUTELY NO WARRANTY.
↦ This is free software, and you are welcome
↦ to redistribute it under certain conditions.
↦ For details, see the file COPYING.
↦ ------------------------------------------------
↦
↦ Linear integer system to solve:
↦
↦ + + +  +  +  +  +  +  +
↦ 0 0 0  0  0  0  0  0  0
↦
↦ 1 1 1 -1 -1 -1  0  0  0 = 0
↦ 1 1 1  0  0  0 -1 -1 -1 = 0
↦ 0 1 1 -1  0  0 -1  0  0 = 0
↦ 1 0 1  0 -1  0  0 -1  0 = 0
↦ 1 1 0  0  0 -1  0  0 -1 = 0
↦ 0 1 1  0 -1  0  0  0 -1 = 0
↦ 1 1 0  0 -1  0 -1  0  0 = 0
↦
↦ Linear integer system of homogeneous equalities to solve:
↦
↦ + + +  +  +  +  +  +  +
↦ 0 0 0  0  0  0  0  0  0
↦
↦ 1 1 1 -1 -1 -1  0  0  0 = 0
↦ 1 1 1  0  0  0 -1 -1 -1 = 0
↦ 0 1 1 -1  0  0 -1  0  0 = 0
↦ 1 0 1  0 -1  0  0 -1  0 = 0
↦ 1 1 0  0  0 -1  0  0 -1 = 0
↦ 0 1 1  0 -1  0  0  0 -1 = 0
↦ 1 1 0  0 -1  0 -1  0  0 = 0
↦
↦
↦ Appending variable 1... Solutions: 5, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 2... Solutions: 4, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 3... Solutions: 3, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 4... Solutions: 4, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 5... Solutions: 4, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 6... Solutions: 7, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 7... Solutions: 6, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 8... Solutions: 5, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 9... Solutions: 5, Step: 0.00 secs, Time: 0.00 secs
↦
↦ Final basis has 1 inhomogeneous, 5 homogeneous and 0 free elements.
↦
↦ 4ti2 Total Time: 0.00 secs
↦ _[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.4.27.3 graver4ti2

Procedure from library `sing4ti2.lib` (see Section D.4.27 [sing4ti2_lib], page 875).

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);
↦ ------------------------------------------------
↦ 4ti2 version 1.3.2, Copyright (C) 2006 4ti2 team.
↦ 4ti2 comes with ABSOLUTELY NO WARRANTY.
↦ This is free software, and you are welcome
↦ to redistribute it under certain conditions.
↦ For details, see the file COPYING.
↦ ------------------------------------------------
↦
↦ Linear integer system to solve:
↦
↦ + + + +
↦ - - - -
↦
↦ 0 1 2 3 = 0
↦ 3 2 1 0 = 0
↦
↦ Linear integer system of homogeneous equalities to solve:
↦
↦ + + + +
↦ - - - -
↦
↦ 0 1 2 3 = 0
↦ 3 2 1 0 = 0
↦
↦
↦ Appending variable 1... Solutions: 4, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 2... Solutions: 4, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 3... Solutions: 8, Step: 0.00 secs, Time: 0.00 secs
↦ Appending variable 4... Solutions: 10, Step: 0.00 secs, Time: 0.00 secs
```

```
↦
↦ Final basis has 1 inhomogeneous, 10 homogeneous and 0 free elements.
↦ Writing 5 vectors to graver file, with respect to symmetry.
↦
↦ 4ti2 Total Time: 0.00 secs
↦ _[1]=-y2+xz
↦ _[2]=-y3+x2w
↦ _[3]=-yz+xw
↦ _[4]=-z2+yw
↦ _[5]=-z3+xw2
```

D.4.28 toric_lib

Library: toric.lib

Purpose: Standard Basis of Toric Ideals

Author: Christine Theis, email: ctheis@math.uni-sb.de

Procedures:

D.4.28.1 toric_ideal

Procedure from library `toric.lib` (see Section D.4.28 [toric_lib], page 880).

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 513.

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
↦    ? leaving toric.lib::toric_ideal
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 513; Section D.4.8 [intprog_lib], page 708; Section D.4.28 [toric_lib], page 880; Section D.4.28.2 [toric_std], page 881.

D.4.28.2 toric_std

Procedure from library `toric.lib` (see Section D.4.28 [toric_lib], page 880).

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 513.

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
// 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(H,xyz);
↦ [1]:
↦    _[1]=x-yz
↦    _[2]=y2-1
↦ [2]:
↦    1
```

See also: Section C.6.1 [Toric ideals], page 513; Section D.4.8 [intprog_lib], page 708; Section D.4.28.1 [toric_ideal], page 880; Section D.4.28 [toric_lib], page 880.

D.5 Singularities

D.5.1 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.5.1.1 resolutiongraph

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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,-2,0,0,0,0,
↦ 0,0,0,1,0,0,-1,0,0,0,
↦ 0,0,0,0,0,0,0,2,1,0,
↦ 1,0,0,0,0,0,0,1,3,1,
↦ 0,0,0,0,0,0,0,0,1,-3
```

See also: Section D.5.1.3 [alexanderpolynomial], page 885; Section D.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.2 totalmultiplicities

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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 multiplicty 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,-2,0,0,0,0,
↦    0,0,0,1,0,0,-1,0,0,0,
↦    0,0,0,0,0,0,0,2,1,0,
↦    1,0,0,0,0,0,0,1,3,1,
↦    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.5.1.3 [alexanderpolynomial], page 885; Section D.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.1.1 [resolutiongraph], page 882.

D.5.1.3 alexanderpolynomial

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.1.1 [resolutiongraph], page 882; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.4 semigroup

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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 thrid 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]:
↦       4,2,2
↦    [2]:
↦       6,3,3
↦ [2]:
↦    [1]:
↦       13,7,6
↦    [2]:
↦       26,13,12
↦    [3]:
↦       12,6,6
↦ [3]:
↦    21,41,20
```

See also: Section D.5.1.1 [resolutiongraph], page 882; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.5 proximitymatrix

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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 multiplicites.
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.5.1.3 [alexanderpolynomial], page 885; Section D.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.6 multseq2charexp

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.1.7 charexp2multseq

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.1.3 [alexanderpolynomial], page 885; Section D.5.9.6 [invariants], page 942; Section D.5.1.1 [resolutiongraph], page 882; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.8 charexp2generators

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.1.3 [alexanderpolynomial], page 885; Section D.5.9.6 [invariants], page 942; Section D.5.1.1 [resolutiongraph], page 882; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.9 charexp2inter

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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 calulated as ...
print(charexp2inter(contact,charexp));
↦       0     4     8
↦       4     0    17
↦       8    17     0
```

See also: Section D.5.9.6 [invariants], page 942; Section D.5.1.1 [resolutiongraph], page 882; Section D.5.1.4 [semigroup], page 886; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.10 charexp2conductor

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.9.6 [invariants], page 942; Section D.5.1.1 [resolutiongraph], page 882; Section D.5.1.4 [semigroup], page 886; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.1.11 charexp2poly

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.1.7 [charexp2multseq], page 890; Section D.5.1.6 [multseq2charexp], page 890.

D.5.1.12 tau_es2

Procedure from library `alexpoly.lib` (see Section D.5.1 [alexpoly_lib], page 882).

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.5.9.2 [develop], page 937; Section D.5.6 [equising_lib], page 913; Section D.5.9.1 [hnexpansion], page 935; Section D.5.6.1 [tau_es], page 914; Section D.5.1.2 [totalmultiplicities], page 883.

D.5.2 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.5.2.1 nashmult

Procedure from library `arcpoint.lib` (see Section D.5.2 [arcpoint_lib], page 893).

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.5.2.2 removepower

Procedure from library `arcpoint.lib` (see Section D.5.2 [arcpoint_lib], page 893).

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.5.2.3 [idealsimplify], page 894.

D.5.2.3 idealsimplify

Procedure from library `arcpoint.lib` (see Section D.5.2 [arcpoint_lib], page 893).

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.5.2.4 equalJinI

Procedure from library `arcpoint.lib` (see Section D.5.2 [arcpoint_lib], page 893).

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.5.2.3 [idealsimplify], page 894.

D.5.3 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:

D.5.3.1 basicinvariants

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.2 classify

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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 @sc{Singular} version 1.0.1. Updates can be found at:
URL=http://www.mathematik.uni-kl.de/~krueger/Singular/

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.5.3.3 corank

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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 occuring 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.5.3.4 Hcode

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.5 init_debug

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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 usefull 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.5.3.6 internalfunctions

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.7 milnorcode

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.8 morsesplit

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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;
↦ // ** ‘r‘ is already global
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.5.3.9 quickclass

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.10 singularity

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

Usage: singularity(t, l); t=string (name of singularity),
l=list of integers/polynomials (indices/parmeters 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.5.3.11 A_L

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.12 normalform

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.13 debug_log

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.3.5 [init_debug], page 898.

D.5.3.14 swap

Procedure from library `classify.lib` (see Section D.5.3 [classify_lib], page 895).

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.5.4 curvepar_lib

Library: space_curve.lib

Author: Viazovska Maryna, email: viazovsk@mathematik.uni-kl.de

Procedures:

D.5.4.1 BlowingUp

Procedure from library `curvepar.lib` (see Section D.5.4 [curvepar_lib], page 903).

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
C[i][1] is an integer j. It shows, which standard chart do we consider. C[i][2] is an irreducible polynomial 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 one 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).

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.5.4.2 CurveRes

Procedure from library `curvepar.lib` (see Section D.5.4 [curvepar_lib], page 903).

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 polynomial g in k[a]. It is a minimal polynomial for the new parameter a. Resolve[i][4] sequence of multiplicities
Resolve[i][5] is a list of integers l. It shows, which standard charts we considered.
Resolve[i][6] HN matrix

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(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(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.5.4.3 CurveParam

Procedure from library `curvepar.lib` (see Section D.5.4 [curvepar_lib], page 903).

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
ring r=0,(x,y,z),dp;
↦ // ** redefining r **
ideal i=x2-y3,z2-y5;
def s=CurveParam(i);
↦ // ** redefining s **
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.5.4.4 WSemigroup

Procedure from library `curvepar.lib` (see Section D.5.4 [curvepar_lib], page 903).

Usage: WSemigroup(X,b0);
X a list of polinomials in one vaiable, 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 greather 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.5.5 deform_lib

Library: deform.lib

Purpose: Miniversal Deformation of Singularities and Modules

Author: Bernd Martin, email: martin@math.tu-cottbus.de

Procedures:

D.5.5.1 versal

Procedure from library `deform.lib` (see Section D.5.5 [deform_lib], page 908).

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.
↦ // ** J is no standard basis
↦
↦
↦ // '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]:
↦    //   characteristic : 0
↦ //   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]:
↦    //   characteristic : 0
↦ //   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]:
↦    //   characteristic : 0
↦ //   number of vars : 4
↦ //        block   1 : ordering ds
↦ //                  : names    A B C D
↦ //        block   2 : ordering C
↦ [4]:
↦    //   characteristic : 0
↦ //   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.5.5.2 mod_versal

Procedure from library `deform.lib` (see Section D.5.5 [deform_lib], page 908).

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.5.5.3 lift_kbase

Procedure from library `deform.lib` (see Section D.5.5 [deform_lib], page 908).

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.5.5.4 lift_rel_kb

Procedure from library `deform.lib` (see Section D.5.5 [deform_lib], page 908).

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.5.6 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

Main procedures: Auxiliary procedure:

D.5.6.1 tau_es

Procedure from library `equising.lib` (see Section D.5.6 [equising_lib], page 913).

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.5.6.2 [esIdeal], page 914; Section D.5.9.6 [invariants], page 942; Section D.5.13.15 [tjurina], page 969.

D.5.6.2 esIdeal

Procedure from library `equising.lib` (see Section D.5.6 [equising_lib], page 913).

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.5.6.3 [esStratum], page 915; Section D.5.6.1 [tau_es], page 914.

D.5.6.3 esStratum

Procedure from library `equising.lib` (see Section D.5.6 [equising_lib], page 913).

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 occured, 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 trival equimultiple sections) and the polynomial p_F=F,
l[2]=1 if some error has occured, 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]=d
↦        _[3]=c
↦        _[4]=-e
↦        _[5]=a
↦ [2]:
↦    0
printlevel=p;
```

See also: Section D.5.6.2 [esIdeal], page 914; Section D.5.6.4 [isEquising], page 918.

D.5.6.4 isEquising

Procedure from library `equising.lib` (see Section D.5.6 [equising_lib], page 913).

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.5.6.5 control_Matrix

Procedure from library `equising.lib` (see Section D.5.6 [equising_lib], page 913).

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.5.9.8 [multsequence], page 944.

D.5.7 gmssing_lib

Library: gaussman.lib

Purpose: Gauss-Manin System of Isolated Singularities

Author: Mathias Schulze, email: mschulze@mathematik.uni-kl.de

Overview: A library to compute invariants related to the the Gauss-Manin system of an isolated hypersurface singularity

Procedures: Bernstein-Sato polynomial See also: Section D.5.8 [gmspoly_lib], page 933; Section D.5.11 [mondromy_lib], page 954; Section D.5.15 [spectrum_lib], page 978.

D.5.7.1 gmsring

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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 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,
↦ 2xy2+5x4,
↦ 5x5-5y5,
↦ 10y6+25x3y4
print(gmsmatrix);
↦ 0,1,x, -2xy,
↦ 1,0,-y,2y2+5x3
print(gmsbasis);
↦ y5,
↦ y4,
↦ y3,
↦ y2,
↦ xy,
↦ y,
↦ x4,
↦ x3,
↦ x2,
↦ x,
↦ 1
gmsmaxdeg;
↦ 1
```

D.5.7.2 gmsnf

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.3 gmscoeffs

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.4 bernstein

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.5 monodromy

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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 638; Section D.5.11 [mondromy_lib], page 954.

D.5.7.6 spectrum

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.15 [spectrum_lib], page 978.

D.5.7.7 sppairs

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.15 [spectrum_lib], page 978.

D.5.7.8 vfilt

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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]=2xy2+5x4
↦    _[3]=5x5-5y5
↦    _[4]=10y6+25x3y4
```

See also: Section D.5.15 [spectrum_lib], page 978.

D.5.7.9 vwfilt

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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]=2xy2+5x4
↦     _[3]=5x5-5y5
↦     _[4]=10y6+25x3y4
```

See also: Section D.5.15 [spectrum_lib], page 978.

D.5.7.10 tmatrix

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.11 endvfilt

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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]=2xy2+5x4
↦    _[3]=5x5-5y5
↦    _[4]=10y6+25x3y4
```

D.5.7.12 sppnf

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

Usage: sppnf(list(a,w[,m])); ideal a, intvec w, intvec m

Assume: ncols(e)==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.5.7.13 sppprint

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.14 spadd

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.15 spsub

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.16 spmul

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.17 spissemicont

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.18 spsemicont

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.19 spmilnor

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.20 spgeomgenus

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.7.21 spgamma

Procedure from library `gmssing.lib` (see Section D.5.7 [gmssing_lib], page 919).

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.5.8 gmspoly_lib

Library: gmspoly.lib

Purpose: Gauss-Manin System of Tame Polynomials

Author: Mathias Schulze, email: mschulze@mathematik.uni-kl.de

Overview: A library to compute invariants related to the Gauss-Manin system of a cohomologically tame polynomial

Procedures: See also: Section D.5.7 [gmssing_lib], page 919.

D.5.8.1 isTame

Procedure from library `gmspoly.lib` (see Section D.5.8 [gmspoly_lib], page 933).

Usage: isTame(f); poly f

Assume: basering has no variables named w(1),w(2),...

Return:

int k=
0; if f is not tame
1; if f is tame

Example:

```
LIB "gmspoly.lib";
ring R=0,(x,y),dp;
isTame(x2y+x);
↦ 0
isTame(x3+y3+xy);
↦ 1
```

D.5.8.2 goodBasis

Procedure from library `gmspoly.lib` (see Section D.5.8 [gmspoly_lib], page 933).

Usage: goodBasis(f); poly f

Assume: f is cohomologically tame

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))

Example:

```
LIB "gmspoly.lib";
ring R=0,(x,y,z),dp;
poly f=x+y+z+x2y2z2;
def Rs=goodBasis(f);
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
↦ // ** _ is no standard basis
↦ // ** rk(submod) > rk(mod) ?
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
```

D.5.9 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.

Main procedures: Auxiliary procedures:

D.5.9.1 hnexpansion

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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;
↦ //   characteristic : 0
↦ //   1 parameter    : a
↦ //   minpoly        : (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.5.9.2 [develop], page 937; Section D.5.9.5 [displayHNE], page 941; Section D.5.9.3 [extdevelop], page 939; Section D.5.9.4 [param], page 940.

D.5.9.2 develop

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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+0(t^21),  y(t)= -3t^98+0(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.5.9.5 [displayHNE], page 941; Section D.5.9.3 [extdevelop], page 939; Section D.5.9.1 [hnexpansion], page 935.

D.5.9.3 extdevelop

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.4 [param], page 940.

D.5.9.4 param

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.3 [extdevelop], page 939.

D.5.9.5 displayHNE

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935.

D.5.9.6 invariants

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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] / 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.5.9.2 [develop], page 937; Section D.5.9.7 [displayInvariants], page 944; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.10 [intersection], page 946; Section D.5.9.8 [multsequence], page 944.

D.5.9.7 displayInvariants

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.10 [intersection], page 946; Section D.5.9.6 [invariants], page 942.

D.5.9.8 multsequence

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.9 [displayMultsequence], page 945; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.17 [separateHNE], page 950.

D.5.9.9 displayMultsequence

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.8 [multsequence], page 944; Section D.5.9.17 [separateHNE], page 950.

D.5.9.10 intersection

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.7 [displayInvariants], page 944; Section D.5.9.1 [hnexpansion], page 935.

D.5.9.11 is_irred

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.12 delta

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.4.15.9 [deltaLoc], page 757; Section D.5.9.6 [invariants], page 942.

D.5.9.13 newtonpoly

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.14 is_NND

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.13 [newtonpoly], page 947.

D.5.9.15 stripHNE

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.3 [extdevelop], page 939; Section D.5.9.1 [hnexpansion], page 935.

D.5.9.16 puiseux2generators

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.6 [invariants], page 942.

D.5.9.17 separateHNE

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.2 [develop], page 937; Section D.5.9.9 [displayMultsequence], page 945; Section D.5.9.1 [hnexpansion], page 935; Section D.5.9.8 [multsequence], page 944.

D.5.9.18 squarefree

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.19 [allsquarefree], page 951.

D.5.9.19 allsquarefree

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.9.18 [squarefree], page 950.

D.5.9.20 further_hn_proc

Procedure from library `hnoether.lib` (see Section D.5.9 [hnoether_lib], page 935).

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.5.10 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.5.10.1 KSker

Procedure from library `kskernel.lib` (see Section D.5.10 [kskernel_lib], page 952).

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 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.5.10.2 KSconvert

Procedure from library `kskernel.lib` (see Section D.5.10 [kskernel_lib], page 952).

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 KSring;
```

```
KSconvert(KSkernel);
↦ // ** 'KSring2' is already global
setring 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.5.10.3 KSlinear

Procedure from library `kskernel.lib` (see Section D.5.10 [kskernel_lib], page 952).

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 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.5.10.4 KScoef

Procedure from library `kskernel.lib` (see Section D.5.10 [kskernel_lib], page 952).

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
↦ // ** redefining u
↦ // ** redefining l
↦ // ** redefining 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.5.10.5 StringF

Procedure from library `kskernel.lib` (see Section D.5.10 [kskernel_lib], page 952).

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.5.11 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.5.8 [gmspoly_lib], page 933; Section D.5.7 [gmssing_lib], page 919.

D.5.11.1 detadj

Procedure from library `mondromy.lib` (see Section D.5.11 [mondromy_lib], page 954).

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 sqaure matrix, the list is empty.
If U is a sqaure 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.5.11.2 invunit

Procedure from library `mondromy.lib` (see Section D.5.11 [mondromy_lib], page 954).

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.5.11.3 jacoblift

Procedure from library `mondromy.lib` (see Section D.5.11 [mondromy_lib], page 954).

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.5.11.4 monodromyB

Procedure from library `mondromy.lib` (see Section D.5.11 [mondromy_lib], page 954).

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.5.11.5 H2basis

Procedure from library `mondromy.lib` (see Section D.5.11 [mondromy_lib], page 954).

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.5.12 qhmoduli_lib

Library: qhmoduli.lib

Purpose: Moduli Spaces of Semi-Quasihomogeneous Singularities

Author: Thomas Bayer, email: bayert@in.tum.de

Procedures:

D.5.12.1 ArnoldAction

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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);
↦ type of i: ?unknown type?
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.5.12.2 ModEqn

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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);
↦ type of i: ?unknown type?
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.5.12.3 QuotientEquations

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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.5.12.4 StabEqn

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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' 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. 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);
↦ type of i: ?unknown type?
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.5.12.5 StabEqnId

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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);
↦ type of i: ?unknown type?
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.5.12.6 StabOrder

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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.5.12.7 UpperMonomials

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

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.5.12.8 Max

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

Usage: Max(data); intvec/list of integers

Purpose: find the maximal integer contained in 'data'

Return: list

Assume: 'data' contians only integers and is not empty

Example:

```
LIB "qhmoduli.lib";
Max(list(1,2,3));
↦ 3
```

D.5.12.9 Min

Procedure from library `qhmoduli.lib` (see Section D.5.12 [qhmoduli_lib], page 957).

Usage: Min(data); intvec/list of integers

Purpose: find the minimal integer contained in 'data'

Return: list

Assume: 'data' contians only integers and is not empty

Example:

```
LIB "qhmoduli.lib";
Min(intvec(1,2,3));
↦ 1
```

D.5.13 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.5.13.1 codim

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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,z),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)
↦ -1
```

D.5.13.2 deform

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.3 dim_slocus

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.4 is_active

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.5 is_ci

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.6 is_is

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.7 is_reg

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.8 is_regs

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.9 locstd

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.10 milnor

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.11 nf_icis

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.12 slocus

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

Usage: slocus(i); i ideal

Return: ideal of singular locus of i

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.5.13.13 qhspectrum

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.14 Tjurina

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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,
↦ 6y6,
↦ 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
↦ //         2 t^0
↦ //        -3 t^1
↦ //        -3 t^2
↦ //         7 t^3
↦ //        -3 t^4
↦
↦ //         2 t^0
↦ //         3 t^1
↦ // dimension (local)   = 0
↦ // multiplicity = 5
printlevel = p;
```

D.5.13.15 tjurina

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.16 T_1

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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(L[2]*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(L[3])*L[2]);  // should be 0 (mod i)
↦ xz,0, -xy,-yz,0,  0,
↦ 0, yz,0,  yz, -xy,-xz
printlevel = p;
```

D.5.13.17 T_2

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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);"";
↦ //         1 t^0
↦ //        -1 t^2
↦ //        -1 t^3
↦ //         1 t^5
↦
↦ //         1 t^0
↦ //         2 t^1
↦ //         2 t^2
↦ //         1 t^3
```

```
↦ // 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.5.13.18 T_12

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.13.19 tangentcone

Procedure from library `sing.lib` (see Section D.5.13 [sing_lib], page 961).

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.5.14 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.5.14.1 isCMcod2

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.2 CMtype

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.3 matrixT1

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.4 semiCMcod2

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.5 discr

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.6 qhmatrix

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.7 relweight

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.8 posweight

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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.5.14.9 KSpencerKernel

Procedure from library `spcurve.lib` (see Section D.5.14 [spcurve_lib], page 973).

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;
↦ //   characteristic : 0
↦ //   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.5.15 spectrum_lib

Library: spectrum.lib

Purpose: Singularity Spectrum for Nondegenerate Singularities

Author: S. Endrass

Procedures:

D.5.15.1 spectrumnd

Procedure from library `spectrum.lib` (see Section D.5.15 [spectrum_lib], page 978).

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 Invariant theory

D.6.1 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..

Main procedures: Auxiliary procedures:

D.6.1.1 invariant_ring

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.2 invariant_ring_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.3 primary_invariants

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.4 primary_invariants_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.5 invariant_algebra_reynolds

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.6 [invariant_algebra_perm], page 983.

D.6.1.6 invariant_algebra_perm

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.5 [invariant_algebra_reynolds], page 982.

D.6.1.7 cyclotomic

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.8 group_reynolds

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.9 molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: molien(G1,G2,...[,ringname,lcm,flags]);
G1,G2,...: nxn <matrices>, all elements of a finite matrix group, ringname: a <string> giving a name for 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 'ringname' 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;
string newring="alksdfjlaskdjf";
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
list L=group_reynolds(A);
molien(L[2..size(L)],newring);
setring alksdfjlaskdjf;
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
setring S;
kill alksdfjlaskdjf;
```

D.6.1.10 reynolds_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: reynolds_molien(G1,G2,...[,ringname,flags]);
G1,G2,...: nxn <matrices> generating a finite matrix group, ringname: a <string> giving a name for 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 'ringname' 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;
string newring="Qadjoint";
matrix A[3][3]=0,1,0,-1,0,0,0,0,-1;
matrix REY=reynolds_molien(A,newring);
print(REY);
↦ y, -x,-z,
↦ -x,-y,z,
↦ -y,x, -z,
↦ x, y, z
setring Qadjoint;
print(M);
↦ x3+x2-x+1,-x7+x6+x5-x4+x3-x2-x+1
setring S;
kill Qadjoint;
```

D.6.1.11 partial_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.12 evaluate_reynolds

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.13 invariant_basis

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: invariant_basis(g,G1,G2,...);
g: an <int> indicating of which degree (>0) the homogeneous basis shoud 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.6.1.14 invariant_basis_reynolds

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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 shoud 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.6.1.15 primary_char0

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.16 primary_charp

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: primary_charp(REY,ringname[,v]);
REY: a <matrix> representing the Reynolds operator, ringname: a <string> giving the name of a 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);
string newring="alskdfj";
molien(L[2..size(L)],newring);
matrix P=primary_charp(L[1],newring);
kill `newring`;
print(P);
↦ z2,x2+y2,x2y2
```

D.6.1.17 primary_char0_no_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.18 primary_charp_no_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.19 primary_charp_without

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.20 primary_char0_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.21 primary_charp_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: primary_charp_random(REY,ringname,r[,v]);
REY: a <matrix> representing the Reynolds operator, ringname: a <string> giving the name of a 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);
string newring="alskdfj";
molien(L[2..size(L)],newring);
matrix P=primary_charp_random(L[1],newring,1);
kill `newring`;
print(P);
↦ z2,x2+y2,x4+y4-z4
```

D.6.1.22 primary_char0_no_molien_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.23 primary_charp_no_molien_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.24 primary_charp_without_random

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.25 power_products

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.26 secondary_char0

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.27 [irred_secondary_char0], page 997.

D.6.1.27 irred_secondary_char0

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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 paramter "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] = 0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,
matrix A2[6][6] = 0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,
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.6.1.26 [secondary_char0], page 995.

D.6.1.28 secondary_charp

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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;
ringname: a <string> giving the name of a 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();
'ringname' is the name of a 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.6.1.29 secondary_no_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.31 [secondary_and_irreducibles_no_molien], page 1001.

D.6.1.30 irred_secondary_no_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.27 [irred_secondary_char0], page 997.

D.6.1.31 secondary_and_irreducibles_no_molien

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.29 [secondary_no_molien], page 999.

D.6.1.32 secondary_not_cohen_macaulay

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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.6.1.33 orbit_variety

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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 's')

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;
string newring="E";
```

```
orbit_variety(F,newring);
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;
↦ //   characteristic : 0
↦ //   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.6.1.34 rel_orbit_variety

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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 occuring 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 654).
For the case of optional string s, the function is equivalent to Section D.6.1.35 [relative_orbit_variety], page 1004.

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;
↦ //   characteristic : 0
↦ //   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.6.1.35 [relative_orbit_variety], page 1004.

D.6.1.35 relative_orbit_variety

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

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,
s: a <string> giving a name for a new ring

Return: The procedure ends with a new ring named s.
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.6.1.34 [rel_orbit_variety], page 1003), 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;
↦ //   characteristic : 0
↦ //   number of vars : 3
↦ //        block   1 : ordering lp
↦ //                  : names    y(1) y(2) y(3)
↦ //        block   2 : ordering C
```

See also: Section D.6.1.34 [rel_orbit_variety], page 1003.

D.6.1.36 image_of_variety

Procedure from library `finvar.lib` (see Section D.6.1 [finvar_lib], page 979).

Usage: image_of_variety(I,F);
I: an arbitray <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.6.2 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.6.2.1 invariantRing

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.2 derivate

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.3 actionIsProper

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.4 reduction

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.5 completeReduction

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.6 localInvar

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.7 furtherInvar

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.2.8 sortier

Procedure from library `ainvar.lib` (see Section D.6.2 [ainvar_lib], page 1005).

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.6.3 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 semi-quasihomogenous singularities and an implementation in Singular'

Procedures: See also: Section D.5.12 [qhmoduli_lib], page 957; Section D.7.5 [zeroset_lib], page 1049.

D.6.3.1 HilbertSeries

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.2 HilbertWeights

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.3 ImageVariety

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.4 ImageGroup

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.5 InvariantRing

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.6 InvariantQ

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.7 LinearizeAction

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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;
↦ //   characteristic : 0
↦ //   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.6.3.8 LinearActionQ

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.9 LinearCombinationQ

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.10 MinimalDecomposition

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.11 NullCone

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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 homogenous, 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.6.3.12 ReynoldsImage

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.13 ReynoldsOperator

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.3.14 SimplifyIdeal

Procedure from library `rinvar.lib` (see Section D.6.3 [rinvar_lib], page 1010).

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.6.4 stratify_lib

Library: stratify.lib

Purpose: Algorithmic Stratification for Unipotent Group-Actions

Author: Anne Fruehbis-Krueger, anne@mathematik.uni-kl.de

Procedures:

D.6.4.1 prepMat

Procedure from library `stratify.lib` (see Section D.6.4 [stratify_lib], page 1017).

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);
intvec wr=1,3,5;
intvec ws=2,4;
int step=2;
prepMat(M,wr,ws,step);
```

D.6.4.2 stratify

Procedure from library `stratify.lib` (see Section D.6.4 [stratify_lib], page 1017).

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);
```

D.7 Symbolic-numerical solving

D.7.1 presolve_lib

Library: presolve.lib

Purpose: Pre-Solving of Polynomial Equations

Author: Gert-Martin Greuel, email: greuel@mathematik.uni-kl.de,

Procedures:

D.7.1.1 degreepart

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.1.2 elimlinearpart

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.1.3 elimpart

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.1.4 elimpartanyr

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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 uses `execute` to create 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.7.1.5 fastelim

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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]=f2+w2-1
↦ _[2]=z2+v2+c2-1
↦ _[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.7.1.6 findvars

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

Usage: findvars(id [,any]); id=poly/ideal/vector/module/matrix, any=any type

Return: if no second argument is present: ideal of variables occuring in id,
if a second argument is given (of any type): list L with 4 entries:

L[1]: ideal of variables occuring in id
L[2]: intvec of variables occuring in id
L[3]: ideal of variables not occuring in id
L[4]: intvec of variables not occuring 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]=f
↦ _[2]=x
↦ _[3]=t
↦ _[4]=w
↦ _[5]=a
findvars(i,1);
↦ [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
```

D.7.1.7 hilbvec

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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: intvec 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.7.1.8 linearpart

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.1.9 tolessvars

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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: (0),(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.7.1.10 solvelinearpart

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.1.11 sortandmap

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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 occuring in pi is sorted w.r.t. its complexity in id,
- ni controls the sorting in i-th block (= vars occuring 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 occuring 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: (32003),(y,z,x),(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]=z2+zx
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: (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.7.1.12 sortvars

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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 occuring 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 occuring 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]=y
↦    _[2]=x
↦    _[3]=w
↦    _[4]=z
↦ [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.7.1.13 valvars

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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 occuring 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 occuring 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 occuring 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]:
↦    3,1,2,4,5
↦ [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.7.1.14 idealSplit

Procedure from library `presolve.lib` (see Section D.7.1 [presolve_lib], page 1018).

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.7.2 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.7.2.1 laguerre_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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:  ring lagc=(complex,30,30),x,lp;
↦ //         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.7.2.2 solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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.7.2.3 ures_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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.7.2.4 mp_res_mat

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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.7.2.5 interpolate

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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 5.1.145 [vandermonde], page 232.

D.7.2.6 fglm_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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),lp;
// 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]:
↦       -2.8284271247
↦    [2]:
↦       -1.4142135624
↦ [3]:
↦    [1]:
↦       2.8284271247
↦    [2]:
↦       1.4142135624
↦ [4]:
↦    [1]:
↦       -1
↦    [2]:
↦       3
```

D.7.2.7 lex_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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]:
↦       -2.8284271247
↦    [2]:
↦       -1.4142135624
↦ [3]:
↦    [1]:
↦       2.8284271247
↦    [2]:
↦       1.4142135624
↦ [4]:
↦    [1]:
↦       -1
↦    [2]:
↦       3
```

D.7.2.8 simplexOut

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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 5.1.124 [simplex], page 214.

D.7.2.9 triangLf_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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]:
↦       -2.8284271247
↦    [2]:
↦       -1.4142135624
↦ [4]:
↦    [1]:
↦       2.8284271247
↦    [2]:
↦       1.4142135624
```

D.7.2.10 triangM_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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]:
↦       -2.8284271247
↦    [2]:
↦       -1.4142135624
↦ [3]:
↦    [1]:
↦       2.8284271247
↦    [2]:
↦       1.4142135624
↦ [4]:
↦    [1]:
↦       -1
↦    [2]:
↦       3
```

D.7.2.11 triangL_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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]:
↦        -2.8284271247
↦     [2]:
↦        -1.4142135624
↦  [3]:
↦     [1]:
↦        2.8284271247
↦     [2]:
↦        1.4142135624
↦  [4]:
↦     [1]:
↦        -1
↦     [2]:
↦        3
```

D.7.2.12 triang_solve

Procedure from library `solve.lib` (see Section D.7.2 [solve_lib], page 1033).

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]:
↦        -2.8284271247
↦     [2]:
↦        -1.4142135624
↦ [4]:
↦     [1]:
↦        2.8284271247
↦     [2]:
↦        1.4142135624
```

D.7.3 triang_lib

Library: triang.lib

Purpose: Decompose Zero-dimensional Ideals into Triangular Sets

Author: D. Hillebrand

Procedures:

D.7.3.1 triangL

Procedure from library `triang.lib` (see Section D.7.3 [triang_lib], page 1046).

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)));
```

D.7.3.2 triangLfak

Procedure from library `triang.lib` (see Section D.7.3 [triang_lib], page 1046).

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)));
```

D.7.3.3 triangM

Procedure from library `triang.lib` (see Section D.7.3 [triang_lib], page 1046).

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);
```

D.7.3.4 triangMH

Procedure from library `triang.lib` (see Section D.7.3 [triang_lib], page 1046).

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)));
```

D.7.4 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.7.4.1 nt_solve

Procedure from library `ntsolve.lib` (see Section D.7.4 [ntsolve_lib], page 1048).

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]=0.869810458155005508200802475093971033553 7
↦ sol[2]=2.821577445750324600849626251771718236940 9
↦ sol[3]=1.132312008466417990006094015711266871731 8
↦ sol[4]=-1.413071026406678849397999475590194239628
```

D.7.4.2 triMNewton

Procedure from library `ntsolve.lib` (see Section D.7.4 [ntsolve_lib], page 1048).

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]=-2.000000000042265738880279143423
↦ l[2]=1.414213562373095048801688724 21
↦ l[3]=1
```

D.7.5 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 @sc{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: Auxiliary procedures:

D.7.5.1 Quotient

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

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.7.5.2 remainder

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

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.7.5.3 roots

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: roots(f); where f is a polynomial

Purpose: compute all roots of f in a finite extension of the ground field without multiplicities.

Return: ring, a polynomial ring over an extension field of the ground field, 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.7.5.4 sqfrNorm

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: sqfrNorm(f); where f is a polynomial

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.7.5.5 zeroSet

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: zeroSet(I [,opt]); I=ideal, opt=integer

Purpose: compute the zero-set of the zero-dim. ideal I, in a finite extension of the ground field.

Return: ring, a polynomial ring over an extension field of the ground field, 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.7.5.6 egcdMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: egcdMain(f, g); where f,g are polynomials

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.7.5.7 factorMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

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.7.5.8 invertNumberMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: invertNumberMain(f); where f is a polynomial

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.7.5.9 quotientMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: quotientMain(f, g); where f,g are polynomials

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.7.5.10 remainderMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: remainderMain(f, g); where f,g are polynomials

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.7.5.11 rootsMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: rootsMain(f); where f is a polynomial

Purpose: compute all roots of f in a finite extension of the ground field 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.7.5.12 sqfrNormMain

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

Usage: sqfrNorm(f); where f is a polynomial

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.7.5.13 containedQ

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

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.7.5.14 sameQ

Procedure from library `zeroset.lib` (see Section D.7.5 [zeroset_lib], page 1049).

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.7.6 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.7.6.1 signcnd

Procedure from library `signcond.lib` (see Section D.7.6 [signcond_lib], page 1055).

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.7.6.3 [firstoct], page 1057; Section D.7.6.2 [psigncnd], page 1056.

D.7.6.2 psigncnd

Procedure from library `signcond.lib` (see Section D.7.6 [signcond_lib], page 1055).

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.7.6.1 [signcnd], page 1055.

D.7.6.3 firstoct

Procedure from library `signcond.lib` (see Section D.7.6 [signcond_lib], page 1055).

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.7.6.1 [signcnd], page 1055.

D.8 Visualization

D.8.1 graphics_lib

Library: graphics.lib

Purpose: Procedures to use Graphics with Mathematica

Author: Christian Gorzel, gorzelc@math.uni-muenster.de

Procedures:

D.8.1.1 staircase

Procedure from library `graphics.lib` (see Section D.8.1 [graphics_lib], page 1057).

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));
ring r1 = 0,(x,y),dp;
ideal I = fetch(r0,I);
staircase("",std(I));
ring r2 = 0,(x,y),wp(2,3);
ideal I = fetch(r0,I);
staircase("",std(I));
// Paste the output into a Mathematica notebook
// active evalutation of the cell with SHIFT RETURN
```

D.8.1.2 mathinit

Procedure from library `graphics.lib` (see Section D.8.1 [graphics_lib], page 1057).

Usage: mathinit();

Return: initializing string for loading Mathematica's ImplicitPlot

Example:

```
LIB "graphics.lib";
mathinit();
// Paste the output into a Mathematica notebook
// active evalutation of the cell with SHIFT RETURN
```

D.8.1.3 mplot

Procedure from library `graphics.lib` (see Section D.8.1 [graphics_lib], page 1057).

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");
// Paste the output into a Mathematica notebook
// active evalutation of the cell with SHIFT RETURN
// --------- space curves --------------
I = x3,-1/10x3+x2,x2;
mplot("",I);
// Paste the output into a Mathematica notebook
// active evalutation 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");
// Paste the output into a Mathematica notebook
```

```
// active evalutation of the cell with SHIFT RETURN
kill rr0,rr1;
```

D.8.2 latex_lib

Library: latex.lib

Purpose: Typesetting of Singular-Objects in LaTeX2e

Author: Christian Gorzel, gorzelc@math.uni-muenster.de

Procedures:

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

D.8.2.1 closetex

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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.8.2.2 opentex

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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.8.2.3 tex

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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 pdfTeXk, Version 3.1415926-1.40.9 (Web2C 7.5.7)
↦  %&-line parsing enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2005/12/01>
↦ Babel <v3.8l> and hyphenation patterns for english, usenglishmax, dumylan\
   g, noh
↦ yphenation, german-x-2008-06-18, ngerman-x-2008-06-18, german, ngerman, l\
   oaded.
↦
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2005/09/16 v1.4f 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.8.2.4 texdemo

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

Usage: texdemo();

Return: nothing; generates a LaTeX2e file called `texlibdemo.tex` explaining the features of `latex.lib` and its global variables.

Note: this procedure may take some time.

D.8.2.5 texfactorize

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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);
↦ $-2\cdot y^{13}\cdot (x+y+1)^{2}\cdot (-x+y)\cdot x^{3}$
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);
↦ $(a+3)\cdot x^{9}\cdot (x+1)^{3}\cdot (x-1)$
```

D.8.2.6 texmap

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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;
↦ // ** ‘@r1‘ is already global
ring r2=0,(u,v),dp;
map @phi =(@r1,u2,uv -v,v2); export @phi;
↦ // ** ‘@phi‘ is already global
list TeXreplace;
TeXreplace[1] = list("@phi","\\phi");    // @phi --> \phi
```

```
export TeXreplace;
↦ // ** ‘TeXreplace‘ is already global
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
↦ // ** ‘TeXaligned‘ is already global
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.8.2.7 texname

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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.8.2.8 texobj

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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)$$
↦
//  --------------  variation of typesetting for ideals ----------
int TeXaligned = 1; export TeXaligned;
↦ // ** 'TeXaligned' is already global
string TeXbrack = "<"; export TeXbrack;
↦ // ** 'TeXbrack' is already global
texobj("",G);
↦ $\left<5x^{4}y^{3}+3y^{4}+2y^{2},3x^{5}y^{2}+12xy^{3}+4xy+6y^{5}\right>$
↦
kill TeXaligned, TeXbrack;
//  --------------  typesetting for matrices ----------
matrix J = jacob(G);
texobj("",J);
↦ $$\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.8.2.9 texpoly

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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);
↦ $-\frac{2a^{2}-b^{3}+2}{a}x^{2}y^{4}z^{5}+(a^{2}+1)x+(a+1)$
texpoly("",1/(a2+2)*x+2/b);
↦ $\frac{1}{a^{2}+2}x+\frac{2}{b}$
```

D.8.2.10 texproc

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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`. `texproc` cannot be applied to itself correctly.

Example:

```
LIB "latex.lib";
proc exp(int i,int j,list #)
{ string s;
if (size(#))
{
for(i;i<=j;i++)
{ s = s + string(j) + string(#); }
}
return(s);
}
export exp;
↦ // ** ‘exp‘ is already global
texproc("","exp");
↦ \begin{verbatim}
↦ proc exp(int i,int j,list #)
↦ {
↦  string s;
↦ if (size(#))
↦ {
↦ for(i;i<=j;i++)
↦ { s = s + string(j) + string(#); }
↦ }
```

```
↦ return(s);
↦
↦ }
↦ \end{verbatim}
↦
kill exp;
```

D.8.2.11 texring

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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 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.8.2.12 rmx

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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 pdfTeXk, Version 3.1415926-1.40.9 (Web2C 7.5.7)
↦  %&-line parsing enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2005/12/01>
↦ Babel <v3.8l> and hyphenation patterns for english, usenglishmax, dumylan\
   g, noh
↦ yphenation, german-x-2008-06-18, ngerman-x-2008-06-18, german, ngerman, l\
   oaded.
↦
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2005/09/16 v1.4f 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.8.2.13 xdvi

Procedure from library `latex.lib` (see Section D.8.2 [latex_lib], page 1059).

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 pdfTeXk, Version 3.1415926-1.40.9 (Web2C 7.5.7)
↦  %&-line parsing enabled.
↦ entering extended mode
↦ (./exp001.tex
↦ LaTeX2e <2005/12/01>
↦ Babel <v3.8l> and hyphenation patterns for english, usenglishmax, dumylan\
   g, noh
↦ yphenation, german-x-2008-06-18, ngerman-x-2008-06-18, german, ngerman, l\
   oaded.
↦
↦ (/usr/share/texmf-dist/tex/latex/base/article.cls
↦ Document Class: article 2005/09/16 v1.4f 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.8.3 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 xv.
Input data is assumed to originate from resolve.lib and reszeta.lib

Procedures:

D.8.3.1 InterDiv

Procedure from library `resgraph.lib` (see Section D.8.3 [resgraph_lib], page 1069).

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
'xv' (X11/Xorg package) and 'dot' (Graphviz package) need to
be in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.8.3.2 ResTree

Procedure from library `resgraph.lib` (see Section D.8.3 [resgraph_lib], page 1069).

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
'xv' (X11/Xorg package) and 'dot' (Graphviz package) need to
be in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.8.3.3 finalCharts

Procedure from library `resgraph.lib` (see Section D.8.3 [resgraph_lib], page 1069).

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 'xv' (X11/Xorg package) need to be
in the standard search PATH

Return: nothing, only generating graphics output in separate window

D.8.4 surf_lib

Library: surf.lib

Purpose: Procedures for Graphics with Surf

Author: Hans Schoenemann

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://www.mathematik.uni-kl.de/pub/Math/Singular/utils/`. The procedure surfer requires the program `surfer` to be installed. You can download `surfer` from `http://www.imaginary2008.de/surfer.imaginary2008.de`
Under Windows, version 159 or newer of `surfer` is required.

Procedures: See also: Section D.8.5 [surfex_lib], page 1072.

D.8.4.1 plot

Procedure from library `surf.lib` (see Section D.8.4 [surf_lib], page 1070).

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.8.4.2 surfer

Procedure from library `surf.lib` (see Section D.8.4 [surf_lib], page 1070).

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.8.5 surfex_lib

Library: surfex.lib

Purpose: Procedures for visualizing and rotating surfaces.
It is still an alpha version (see http://www.AlgebraicSurface.net)

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: 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.8.4 [surf_lib], page 1070.

D.8.5.1 plotRotated

Procedure from library `surfex.lib` (see Section D.8.5 [surfex_lib], page 1072).

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.8.5.2 plotRot

Procedure from library `surfex.lib` (see Section D.8.5 [surfex_lib], page 1072).

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.8.5.3 plotRotatedList

Procedure from library `surfex.lib` (see Section D.8.5 [surfex_lib], page 1072).

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.8.5.4 plotRotatedDirect

Procedure from library `surfex.lib` (see Section D.8.5 [surfex_lib], page 1072).

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.8.5.5 plotRotatedListFromSpecifyList

Procedure from library `surfex.lib` (see Section D.8.5 [surfex_lib], page 1072).

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.9 Coding theory

D.9.1 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.

Main procedures: Auxiliary procedures:

D.9.1.1 Adj_div

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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]=y2+y+1
↦    _[2]=x+1
↦ [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]:
↦    2,1
↦ [2]:
↦    1,1
↦ [3]:
↦    1,2
↦ [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];
↦ //   characteristic : 2
↦ //   1 parameter    : a
↦ //   minpoly        : ...
↦ //   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.9.1.2 [NSplaces], page 1078; Section D.9.1.10 [closed_points], page 1087.

D.9.1.2 NSplaces

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.1 [Adj_div], page 1075 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+y2+y+1
↦       _[2]=x+y2+y+1
↦    [2]:
↦       13
↦ [3]:
```

```
↦     [1]:
↦        _[1]=y4+y3+1
↦        _[2]=x+y3+y
↦     [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+a2)
↦ [2]:
↦     (a2+a+1)
↦ [3]:
↦     1
printlevel=plevel;
```

See also: Section D.9.1.1 [Adj_div], page 1075; Section D.9.1.10 [closed_points], page 1087.

D.9.1.3 BrillNoether

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.1 [Adj_div], page 1075; Section D.9.1.2 [NSplaces], page 1078; Section D.9.1.4 [Weierstrass], page 1081.

D.9.1.4 Weierstrass

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.1 [Adj_div], page 1075; Section D.9.1.3 [BrillNoether], page 1080; Section D.9.1.2 [NSplaces], page 1078.

D.9.1.5 extcurve

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.1 [Adj_div], page 1075):

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+a2+a+1)
↦ [2]:
↦    (a2+a)
↦ [3]:
↦    1
printlevel=plevel;
```

See also: Section D.9.1.6 [AGcode_L], page 1083; Section D.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.1 [Adj_div], page 1075; Section D.9.1.2 [NSplaces], page 1078; Section D.9.1.10 [closed_points], page 1087.

D.9.1.6 AGcode_L

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.5 [extcurve], page 1082).
The intvec G represents a rational divisor (see Section D.9.1.3 [BrillNoether], page 1080 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,(a),  (a+1),1,  1,    (a+1),(a),
↦ 1,0,(a),  (a+1),(a),(a+1),(a),  (a+1),
↦ 1,1,1,    1,    1,  1,    1,    1,
↦ 0,0,(a+1),(a),  1,  1,    (a),  (a+1),
↦ 0,0,(a+1),(a),  (a),(a+1),1,    1
printlevel=plevel;
```

See also: Section D.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.1 [Adj_div], page 1075; Section D.9.1.3 [BrillNoether], page 1080; Section D.9.1.5 [extcurve], page 1082.

D.9.1.7 AGcode_Omega

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.5 [extcurve], page 1082).
The intvec G represents a rational divisor (see Section D.9.1.3 [BrillNoether], page 1080 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);
↦ 0,    (a),(a),(a),  (a+1),1,0,  0,
↦ (a+1),1,  (a),0,    (a),  0,(a),0,
↦ (a+1),0,  (a),(a+1),(a+1),0,0,  1
printlevel=plevel;
```

See also: Section D.9.1.6 [AGcode_L], page 1083; Section D.9.1.1 [Adj_div], page 1075; Section D.9.1.3 [BrillNoether], page 1080; Section D.9.1.5 [extcurve], page 1082.

D.9.1.8 prepSV

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.3 [BrillNoether], page 1080 for more details).
The intvec D refers to rational places (see Section D.9.1.7 [AGcode_Omega], page 1084 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,(a),  (a+1),1,  1,    (a+1),(a),
↦ 1,0,(a),  (a+1),(a),(a+1),(a),  (a+1),
↦ 1,1,1,    1,    1,  1,    1,    1,
↦ 0,0,(a+1),(a),  1,  1,    (a),  (a+1),
↦ 0,0,(a+1),(a),  (a),(a+1),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.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.9 [decodeSV], page 1087; Section D.9.1.5 [extcurve], page 1082; Section D.9.1.13 [permute_L], page 1090; Section D.9.1.12 [sys_code], page 1089.

D.9.1.9 decodeSV

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.5 [extcurve], page 1082; Section D.9.1.8 [prepSV], page 1085.

D.9.1.10 closed_points

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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]=y2+y+1
↦    _[2]=x+1
↦ [2]:
↦    _[1]=y2+y+1
↦    _[2]=x+y
↦ [3]:
↦    _[1]=y2+y+1
↦    _[2]=x+y+1
↦ [4]:
↦    _[1]=y2+y+1
↦    _[2]=x
↦ [5]:
↦    _[1]=y+1
↦    _[2]=x2+x+1
↦ [6]:
↦    _[1]=y+1
↦    _[2]=x+1
↦ [7]:
↦    _[1]=y+1
↦    _[2]=x
↦ [8]:
↦    _[1]=y
↦    _[2]=x2+x+1
↦ [9]:
↦    _[1]=y
↦    _[2]=x+1
↦ [10]:
↦    _[1]=y
↦    _[2]=x
```

See also: Section D.7.3 [triang_lib], page 1046.

D.9.1.11 dual_code

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.12 sys_code

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.13 [permute_L], page 1090; Section D.9.1.8 [prepSV], page 1085.

D.9.1.13 permute_L

Procedure from library `brnoeth.lib` (see Section D.9.1 [brnoeth_lib], page 1075).

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.9.1.7 [AGcode_Omega], page 1084; Section D.9.1.8 [prepSV], page 1085; Section D.9.1.12 [sys_code], page 1089.

D.9.2 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 517. For the vanishing ideal computation the algorithm of Farr and Gao is implemented.

Main procedures:

D.9.2.1 sysCRHT

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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), containig 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 518.

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
↦ //   characteristic : 2
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   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.9.2.4 [sysBin], page 1097; Section D.9.2.3 [sysNewton], page 1094.

D.9.2.2 sysCRHTMindist

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 518.

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
↦ //   characteristic : 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.9.2.3 sysNewton

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 520.

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
↦ //   characteristic : 2
↦ //   1 parameter    : a
↦ //   minpoly        : 0
↦ //   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.9.2.4 [sysBin], page 1097; Section D.9.2.1 [sysCRHT], page 1091.

D.9.2.4 sysBin

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 elment 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 520.

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.9.2.1 [sysCRHT], page 1091; Section D.9.2.3 [sysNewton], page 1094.

D.9.2.5 encode

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.6 syndrome

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.7 sysQE

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 522.

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.9.2.17 [sysFL], page 1113.

D.9.2.8 errorInsert

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

Usage: errorInsert(y,pos,val); y is matrix, pos,val are list of int's
- y is a (code) word,
- pos = positions where errors occured,
- 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.9.2.9 errorRand

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.10 randomCheck

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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,
↦ 0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1
//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.9.2.11 genMDSMat

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 522.

D.9.2.12 mindist

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.13 decode

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 522 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.9.2.14 decodeRandom

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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:
↦ 0,1,0,0,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 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,
↦ 0,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
↦ 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,
↦ 0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
↦ 1,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
↦ 0,1,1,1,0,1,0,0,1,1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
↦ 0,0,1,0,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
↦ 0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
↦ 0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
↦ 1,0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
↦ 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,
↦ 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,
↦ 0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
↦ 0,0,1,0,0,1,1,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,1,1,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
↦ 1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
↦ 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,
↦ 0,1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
↦ 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,
↦ 0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
↦ 1,0,1,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
↦ 1,0,0,1,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
↦ 1,0,0,1,0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
↦ 1,0,0,0,1,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
↦ 1,0,0,0,1,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
↦ 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.9.2.15 decodeCode

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.16 vanishId

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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.9.2.17 sysFL

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 521.

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.9.2.7 [sysQE], page 1099.

D.9.2.18 decodeRandomFL

Procedure from library `decodegb.lib` (see Section D.9.2 [decodegb_lib], page 1090).

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 pecularities 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
↦ 1,1,0,0,1,0,1,0,0,0,0,0,0,0,1,1,0,0,1,0,1,1,0,1,1,1
↦ 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.10 System and Control theory

D.10.1 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.10.2 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

Main procedures: Component procedures: Auxiliary procedures:

D.10.2.1 control

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.2 controlDim

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.3 autonom

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.4 autonomDim

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.5 leftKernel

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.6 rightKernel

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.7 leftInverse

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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(L*M);
↦ 1
```

D.10.2.8 rightInverse

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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(M*L);
↦ 1
```

D.10.2.9 colrank

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.10 genericity

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.11 canonize

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.12 iostruct

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.13 findTorsion

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.2.14 controlExample

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

Usage: controlExample(s); s a string

Return: ring

Purpose: set up an example from the mini database by initalizing 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.10.2.15 view

Procedure from library `control.lib` (see Section D.10.2 [control_lib], page 1119).

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.10.3 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

Theory: 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.10.2 [control_lib], page 1119.

D.10.3.1 smith

Procedure from library `jacobson.lib` (see Section D.10.3 [jacobson_lib], page 1130).

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.10.3.3 [divideUnits], page 1133; Section D.10.3.2 [jacobson], page 1132.

D.10.3.2 jacobson

Procedure from library `jacobson.lib` (see Section D.10.3 [jacobson_lib], page 1130).

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.10.3.3 [divideUnits], page 1133; Section D.10.3.1 [smith], page 1131.

D.10.3.3 divideUnits

Procedure from library `jacobson.lib` (see Section D.10.3 [jacobson_lib], page 1130).

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 Teaching

The libraries in this section are intended to be used for teaching purposes but not for serious computations.

D.11.1 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.11.1.1 fastExpt

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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";
ring R = 0,x,dp;
fastExpt(2,10,1022);
↦ 1023
```

D.11.1.2 log2

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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";
ring R = 0,x,dp;
log2(1024);
↦ 10
```

D.11.1.3 PerfectPowerTest

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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";
ring R = 0,x,dp;
PerfectPowerTest(887503681);
↦ 0
```

D.11.1.4 wurzel

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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);
↦ 18657101299293613172103220575105621627087185275238394968632787340772624805\
  04411994482808890692946129926517/21359870359209100823950217061695521146025\
  70452235665276994704160782221972578064055002296208693657 6
```

D.11.1.5 euler

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

Usage: euler(r);

Return: number 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";
ring R = 0,x,dp;
euler(99991);
↦ 99990
```

D.11.1.6 coeffmod

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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;
number n=3;
coeffmod(f,n);
↦ x3+x2+2
```

D.11.1.7 powerpolyX

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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;
number q=123;
number n=5;
powerpolyX(q,n,a,r);
↦ 3x5+2x4+x3+x2+1
```

D.11.1.8 ask

Procedure from library `aksaka.lib` (see Section D.11.1 [aksaka_lib], page 1134).

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";
ring R = 0,x,dp;
//ask(100003);
ask(32003);
↦ 1
```

D.11.2 atkins_lib

Library: atkins.lib

Purpose: Procedures for teaching cryptography

Author: Stefan Steidel, steidel@mathematik.uni-kl.de

Note: The library contains auxiliary procedures to compute the elliptic curve primality test of Atkin and the Atkin's Test itself. 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.11.2.1 newTest

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: newTest(L,D);

Return: 1, if D does not already exist in L,
-1, if D does already exist in L

Example:

```
LIB "atkins.lib";
ring r = 0,x,dp;
list L=8976,-223456,556,-778,3,-55603,45,766677;
number D=-55603;
newTest(L,D);
↦ -1
```

D.11.2.2 bubblesort

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: bubblesort(L);

Return: list L, sort in decreasing order

Example:

```
LIB "atkins.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.11.2.3 disc

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: disc(N,k);

Return: list L of negative discriminants D, sorted in decreasing order

Assume: D<0, D kongruent 0 or 1 modulo 4 and |D|<4N

Note: D=b^2-4*a, where 0<=b<=k and intPart((b^2)/4)+1<=a<=k for each b

Example:

```
LIB "atkins.lib";
ring R = 0,x,dp;
disc(2003,50);
↦ [1]:
↦    -3
↦ [2]:
↦    -4
↦ [3]:
↦    -7
↦ [4]:
↦    -8
↦ [5]:
↦    -11
↦ [6]:
↦    -12
↦ [7]:
↦    -15
↦ [8]:
↦    -16
↦ [9]:
↦    -19
↦ [10]:
↦    -20
↦ [11]:
↦    -23
↦ [12]:
↦    -24
↦ [13]:
↦    -27
↦ [14]:
↦    -28
↦ [15]:
↦    -31
↦ [16]:
↦    -32
↦ [17]:
↦    -35
↦ [18]:
↦    -36
↦ [19]:
↦    -39
↦ [20]:
↦    -40
↦ [21]:
↦    -43
↦ [22]:
↦    -44
↦ [23]:
↦    -47
↦ [24]:
↦    -48
↦ [25]:
↦    -51
```

```
↦ [26]:
↦    -52
↦ [27]:
↦    -55
↦ [28]:
↦    -56
↦ [29]:
↦    -59
↦ [30]:
↦    -60
↦ [31]:
↦    -63
↦ [32]:
↦    -64
↦ [33]:
↦    -67
↦ [34]:
↦    -68
↦ [35]:
↦    -71
↦ [36]:
↦    -72
↦ [37]:
↦    -75
↦ [38]:
↦    -76
↦ [39]:
↦    -79
↦ [40]:
↦    -80
↦ [41]:
↦    -83
↦ [42]:
↦    -84
↦ [43]:
↦    -87
↦ [44]:
↦    -88
↦ [45]:
↦    -91
↦ [46]:
↦    -92
↦ [47]:
↦    -95
↦ [48]:
↦    -96
↦ [49]:
↦    -99
↦ [50]:
↦    -100
↦ [51]:
↦    -103
↦ [52]:
```

```
↦    -104
↦ [53]:
↦    -107
↦ [54]:
↦    -108
↦ [55]:
↦    -111
↦ [56]:
↦    -112
↦ [57]:
↦    -115
↦ [58]:
↦    -116
↦ [59]:
↦    -119
↦ [60]:
↦    -120
↦ [61]:
↦    -123
↦ [62]:
↦    -124
↦ [63]:
↦    -127
↦ [64]:
↦    -128
↦ [65]:
↦    -131
↦ [66]:
↦    -132
↦ [67]:
↦    -135
↦ [68]:
↦    -136
↦ [69]:
↦    -139
↦ [70]:
↦    -140
↦ [71]:
↦    -143
↦ [72]:
↦    -144
↦ [73]:
↦    -147
↦ [74]:
↦    -148
↦ [75]:
↦    -151
↦ [76]:
↦    -152
↦ [77]:
↦    -155
↦ [78]:
↦    -156
```

```
↦ [79]:
↦    -159
↦ [80]:
↦    -160
↦ [81]:
↦    -163
↦ [82]:
↦    -164
↦ [83]:
↦    -167
↦ [84]:
↦    -168
↦ [85]:
↦    -171
↦ [86]:
↦    -172
↦ [87]:
↦    -175
↦ [88]:
↦    -176
↦ [89]:
↦    -179
↦ [90]:
↦    -180
↦ [91]:
↦    -183
↦ [92]:
↦    -184
↦ [93]:
↦    -187
↦ [94]:
↦    -188
↦ [95]:
↦    -191
↦ [96]:
↦    -192
↦ [97]:
↦    -195
↦ [98]:
↦    -196
↦ [99]:
↦    -199
↦ [100]:
↦    -200
```

D.11.2.4 Cornacchia

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: Cornacchia(d,p);

Return: x,y such that x^2+d*y^2=p with p prime,
-1, if the Diophantine equation has no solution,
0, if the parameters are wrong selected

Assume: 0<d<p

Example:

```
LIB "atkins.lib";
ring R = 0,x,dp;
Cornacchia(55,9551);
↦ [1]:
↦    16
↦ [2]:
↦    13
```

D.11.2.5 CornacchiaModified

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: CornacchiaModified(D,p);

Return: x,y such that x^2+|D|*y^2=p with p prime,
-1, if the Diophantine equation has no solution,
0, if the parameters are wrong selected

Assume: D<0, D kongruent 0 or 1 modulo 4 and |D|<4p

Example:

```
LIB "atkins.lib";
ring R = 0,x,dp;
CornacchiaModified(-107,1319);
↦ [1]:
↦    51
↦ [2]:
↦    5
```

D.11.2.6 maximum

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: maximum(list L);

Return: the maximal number contained in list L

Example:

```
LIB "atkins.lib";
ring r = 0,x,dp;
list L=465,867,1233,4567,776544,233445,2334,556;
maximum(L);
↦ 776544
```

D.11.2.7 sqr

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Return: the square root of w

Assume: w>=0

Note: k describes the number of decimals being calculated in the real numbers, k, intPart(k/5) are inputs for the procedure "nt_solve"

Example:

```
LIB "atkins.lib";
ring R = (real,60),x,dp;
number ww=28846965010866953572608 1;
sqr(ww,60);
↦ 537093744140.289670042554456783715698474723764341699853789621376194629852
```

D.11.2.8 expo

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: expo(z,k);

Return: e^z to the order k

Note: k describes the number of summands being calculated in the exponential power series

Example:

```
LIB "atkins.lib";
ring r = (real,30),x,dp;
number z=40.35;
expo(z,1000);
↦ 334027593585379682.006907602277432043532129597609
```

D.11.2.9 jOft

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: jOft(t,k);

Return: the j-invariant of t

Assume: t is a complex number with positive imaginary part

Note: k describes the number of summands being calculated in the power series, 10*k is input for the procedure `expo`

Example:

```
LIB "atkins.lib";
ring r = (complex,30,i),x,dp;
number t=(-7+i*sqr(7,250))/2;
jOft(t,50);
↦ -3375
```

D.11.2.10 round

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: round(r);

Return: the nearest number to r out of Z

Assume: r should be a rational or a real number

Example:

```
LIB "atkins.lib";
ring R = (real,50),x,dp;
number r=7357683445788723456321.6788643224;
round(r);
↦ 735768344578872345632 2
```

D.11.2.11 HilbertClassPoly

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

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 "atkins.lib";
ring r = 0,x,dp;
number D=-23;
HilbertClassPoly(D,50);
↦ x3+3491750x2-5151296875x+12771880859375
```

D.11.2.12 rootsModp

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: rootsModp(p,P);

Return: list of roots of the polynomial P modulo p with p prime

Assume: p>=3

Note: this algorithm will be called recursively, and it is understood that all the operations are done in Z/pZ (excepting squareRoot(d,p))

Example:

```
LIB "atkins.lib";
ring r = 0,x,dp;
poly f=x4+2x3-5x2+x;
rootsModp(7,f);
↦ [1]:
↦    0
↦ [2]:
↦    -1
↦ [3]:
↦    2
↦ [4]:
↦    -3
poly g=x5+112x4+655x3+551x2+1129x+831;
rootsModp(1223,g);
↦ [1]:
↦    -33
↦ [2]:
↦    -225
↦ [3]:
↦    206
↦ [4]:
↦    295
↦ [5]:
↦    -355
```

D.11.2.13 wUnit

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Usage: wUnit(D);

Return: the number of roots of unity in the quadratic order of discriminant D

Assume: D<0 a discriminant kongruent to 0 or 1 modulo 4

Example:

```
LIB "atkins.lib";
ring r = 0,x,dp;
number D=-3;
wUnit(D);
↦ 6
```

D.11.2.14 Atkin

Procedure from library `atkins.lib` (see Section D.11.2 [atkins_lib], page 1136).

Return: 1, if N is prime,
-1, if N is not prime,
0, if the algorithm is not applicable, since there are too few discriminants

Assume: N is coprime to 6 and different from 1

Note: K/2 is input for the procedure "disc",
K is input for the procedure "HilbertClassPoly",
B describes the number of recursions being calculated.
The basis of the algorithm is the following theorem:
Let N be an integer coprime to 6 and different from 1 and E be an ellipic curve modulo N.
Assume that we know an integer m and a point P of E(Z/NZ) satisfying the following conditions.
(1) There exists a prime divisor q of m such that q > (4-th root(N)+1)^2.
(2) m*P = O(E) = (0:1:0).
(3) (m/q)*P = (x:y:t) with t element of (Z/NZ)*.
Then N is prime.

Example:

```
LIB "atkins.lib";
ring R = 0,x,dp;
Atkin(7691,100,5);
↦ 1
Atkin(3473,10,2);
↦ -1
printlevel=1;
Atkin(10000079,100,2);
↦ Set i = 0, n = 0 and N(i) = N(0)= 10000079.
↦ pause>
↦ List H of possibly suitable discriminants will be calculated.
↦ H = -3,-4,-7,-8,-11,-12,-15,-16,-19,-20,-23,-24,-27,-28,-31,-32,-35,-36,-\
   39,-40,-43,-44,-47,-48,-51,-52,-55,-56,-59,-60,-63,-64,-67,-68,-71,-72,-7\
   5,-76,-79,-80,-83,-84,-87,-88,-91,-92,-95,-96,-99,-100,-103,-104,-107,-10\
   8,-111,-112,-115,-116,-119,-120,-123,-124,-127,-128,-131,-132,-135,-136,-\
```

```
   139,-140,-143,-144,-147,-148,-151,-152,-155,-156,-159,-160,-163,-164,-167\
   ,-168,-171,-172,-175,-176,-179,-180,-183,-184,-187,-188,-191,-192,-195,-1\
   96,-199,-200
↦ pause>
↦ Next discriminant D will be chosen. D = -3.
↦ pause>
↦ Jacobi(D,N(0)) = -1
↦ pause>
↦ Next discriminant D will be chosen. D = -4.
↦ pause>
↦ Jacobi(D,N(0)) = -1
↦ pause>
↦ Next discriminant D will be chosen. D = -7.
↦ pause>
↦ Jacobi(D,N(0)) = -1
↦ pause>
↦ Next discriminant D will be chosen. D = -8.
↦ pause>
↦ Jacobi(D,N(0)) = -1
↦ pause>
↦ Next discriminant D will be chosen. D = -11.
↦ pause>
↦ The solution (x,y) of the equation x^2+|D|y^2 = 4N(0) is
↦ [1]:
↦    4596
↦ [2]:
↦    1310
↦ pause>
↦ List L2 of possible m = |E(Z/N(0)Z)| will be calculated.
↦ L2 =
↦ [1]:
↦    10004676
↦ [2]:
↦    9995484
↦ pause>
↦ List S of factors of all possible m will be calculated.
↦ S=
↦ [1]:
↦    [1]:
↦       10004676
↦    [2]:
↦       [1]:
↦          11
↦       [2]:
↦          2
↦       [3]:
↦          3
↦       [4]:
↦          75793
↦       [5]:
↦          12
↦ [2]:
↦    [1]:
```

```
↦       9995484
↦    [2]:
↦       [1]:
↦          3
↦       [2]:
↦          832957
↦       [3]:
↦          2
↦       [4]:
↦          3
↦ pause>
↦ Suitable pair (m,q) has been found such that q|m,
↦ q > (4-th root(N(0))+1)^2 and q passes the Miller-Rabin-Test.
↦ m = 10004676,
↦ q = 75793
↦ pause>
↦ The minimal polynomial T of j((D+sqr(D))/2) in Z[X] will be calculated fo\
  r D=-11.
↦ T = x+32768
↦ pause>
↦ Set T = T mod N(0).
↦ T = x+32768
↦ pause>
↦ The zero of T modulo N(0) is
↦ [1]:
↦    -32768
↦ pause>
↦ Choose the zero j = -32768 and set
↦ c = j/(j-1728) mod N(0), a = -3c mod N(0), b = 2c mod N(0).
↦ a = 2374784,
↦ b = 5083530
↦ pause>
↦ g = 3035792
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(0) = 10000079,
↦    a = 2374784,
↦    b = 5083530
↦ will be chosen.
↦ P = (2776172,3288655,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (6511549,5541835,1),
↦ P2 = (6106160,8946469,1)
↦ pause>
↦ Since P1 != (0:1:0), it holds m != |E(Z/N(0)Z)| for the coefficients a = \
  2374784 and b = 5083530.
↦ Therefore choose new coefficients a and b.
↦ pause>
↦ Since D < -4, set a = a*g^2 mod N(0) and b = b*g^3 mod N(0).
↦ a = 1185224,
↦ b = 3258230,
```

```
↦ k = 1
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(0) = 10000079,
↦    a = 1185224,
↦    b = 3258230
↦ will be chosen.
↦ P = (3401270,6866912,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (0,1,0),
↦ P2 = (85781,4173051,1)
↦ pause>
↦ 1. Recursion:
↦
↦ N(0) = 10000079 suffices the conditions of the underlying theorem,
↦ since P1 = (0:1:0) and P2[3] in (Z/N(0)Z)*.
↦
↦ Now check if also the found factor q=75793 suffices these assumptions.
↦ Therefore set i = i+1, N(1) = q = 75793 and restart the algorithm.
↦ pause>
↦ Next discriminant D will be chosen. D = -3.
↦ pause>
↦ The solution (x,y) of the equation x^2+|D|y^2 = 4N(1) is
↦ [1]:
↦    542
↦ [2]:
↦    56
↦ pause>
↦ List L2 of possible m = |E(Z/N(1)Z)| will be calculated.
↦ L2 =
↦ [1]:
↦    76336
↦ [2]:
↦    75252
↦ [3]:
↦    76149
↦ [4]:
↦    75439
↦ [5]:
↦    75981
↦ [6]:
↦    75607
↦ pause>
↦ List S of factors of all possible m will be calculated.
↦ S=
↦ [1]:
↦    [1]:
↦       76336
↦    [2]:
↦       [1]:
↦          13
```

```
↦        [2]:
↦           367
↦        [3]:
↦           2
↦        [4]:
↦           367
↦ [2]:
↦    [1]:
↦       75252
↦    [2]:
↦        [1]:
↦           3
↦        [2]:
↦           6271
↦        [3]:
↦           2
↦        [4]:
↦           12
↦ [3]:
↦    [1]:
↦       76149
↦    [2]:
↦        [1]:
↦           3
↦        [2]:
↦           8461
↦        [3]:
↦           3
↦ [4]:
↦    [1]:
↦       75439
↦    [2]:
↦        [1]:
↦           7
↦        [2]:
↦           13
↦        [3]:
↦           829
↦        [4]:
↦           75439
↦ [5]:
↦    [1]:
↦       75981
↦    [2]:
↦        [1]:
↦           19
↦        [2]:
↦           31
↦        [3]:
↦           43
↦        [4]:
↦           3
↦        [5]:
```

```
↦          75981
↦ [6]:
↦    [1]:
↦       75607
↦    [2]:
↦       [1]:
↦          7
↦       [2]:
↦          1543
↦       [3]:
↦          75607
↦ pause>
↦ Suitable pair (m,q) has been found such that q|m,
↦ q > (4-th root(N(1))+1)^2 and q passes the Miller-Rabin-Test.
↦ m = 76336,
↦ q = 367
↦ pause>
↦ Since D = -3, set a = 0 and b = -1.
↦ pause>
↦ g = 15675
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(1) = 75793,
↦    a = 0,
↦    b = -1
↦ will be chosen.
↦ P = (6739,21159,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (58298,43364,1),
↦ P2 = (75424,29911,1)
↦ pause>
↦ Since P1 != (0:1:0), it holds m != |E(Z/N(1)Z)| for the coefficients a = \
  0 and b = -1.
↦ Therefore choose new coefficients a and b.
↦ pause>
↦ Since D = -3, set b = b*g mod N(1).
↦ a = 0,
↦ b = 60118,
↦ k = 1
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(1) = 75793,
↦    a = 0,
↦    b = 60118
↦ will be chosen.
↦ P = (44443,30044,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (57592,64619,1),
↦ P2 = (65526,31623,1)
```

```
↦ pause>
↦ Since P1 != (0:1:0), it holds m != |E(Z/N(1)Z)| for the coefficients a = \
  0 and b = 60118.
↦ Therefore choose new coefficients a and b.
↦ pause>
↦ Since D = -3, set b = b*g mod N(1).
↦ a = 0,
↦ b = 15281,
↦ k = 2
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(1) = 75793,
↦    a = 0,
↦    b = 15281
↦ will be chosen.
↦ P = (46843,52174,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (49909,3260,1),
↦ P2 = (37869,50319,1)
↦ pause>
↦ Since P1 != (0:1:0), it holds m != |E(Z/N(1)Z)| for the coefficients a = \
  0 and b = 15281.
↦ Therefore choose new coefficients a and b.
↦ pause>
↦ Since D = -3, set b = b*g mod N(1).
↦ a = 0,
↦ b = 23795,
↦ k = 3
↦ pause>
↦ A random point P on the elliptic curve corresponding
↦ to the equation y^2 = x^3+ax+b for
↦ N(1) = 75793,
↦    a = 0,
↦    b = 23795
↦ will be chosen.
↦ P = (32297,22617,1)
↦ pause>
↦ The points P2 = (m/q)*P and P1 = q*P2 on the curve will be calculated.
↦ P1 = (0,1,0),
↦ P2 = (17725,53258,1)
↦ pause>
↦ 2. Recursion:
↦
↦ N(1) = 75793 suffices the conditions of the underlying theorem,
↦ since P1 = (0:1:0) and P2[3] in (Z/N(1)Z)*.
↦
↦ Now check if also the found factor q=367 suffices these assumptions.
↦ Therefore set i = i+1, N(2) = q = 367 and restart the algorithm.
↦ pause>
↦ 1
↦ >
```

D.11.3 crypto_lib

Library: crypto.lib

Purpose: Procedures for teaching cryptography

Author: Gerhard Pfister, pfister@mathematik.uni-kl.de

Note: 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.11.3.1 decimal

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: decimal(s); s = string

Return: the (decimal) number corresponding to the hexadecimal number s

Example:

```
LIB "crypto.lib";
string s  ="8edfe37dae96cfd2466d77d3884d4196";
decimal(s);
↦ 18991287166544437571634062839566861967 0
```

D.11.3.2 exgcdN

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: exgcdN(a,n);

Return: a list s,t,d of numbers satisfying d=gcd(a,n)=s*a+t*n

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
exgcdN(24,15);
↦ [1]:
↦    2
↦ [2]:
↦    -3
↦ [3]:
↦    3
```

D.11.3.3 eexgcdN

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
eexgcdN(list(24,15,21));
↦ [1]:
↦    2
↦ [2]:
↦    -3
↦ [3]:
↦    0
↦ [4]:
↦    3
```

D.11.3.4 gcdN

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: gcdN(a,b);

Return: gcd(a,b)

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
gcdN(24,15);
↦ 3
```

D.11.3.5 lcmN

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: lcmN(a,b);

Return: lcm(a,b);

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
lcmN(24,15);
↦ 120
```

D.11.3.6 powerN

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: powerN(m,d,n);

Return: m^d mod n

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
powerN(24,15,7);
↦ 6
```

D.11.3.7 chineseRem

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: chineseRem(T,L);

Return: x such that x = T[i] mod L[i]

Note: chinese remainder theorem

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
chineseRem(list(24,15,7),list(2,3,5));
↦ 12
```

D.11.3.8 Jacobi

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
Jacobi(13580555397810650806,5792543);
↦ 1
```

D.11.3.9 primList

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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.11.3.10 primL

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
primL(20);
↦ [1]:
↦    2
↦ [2]:
↦    3
↦ [3]:
↦    5
```

D.11.3.11 intPart

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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.11.3.12 intRoot

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

Usage: intRoot(m);

Return: the integral part of the square root of m

Example:

```
LIB "crypto.lib";
ring R = 0,x,dp;
intRoot(20);
↦ 4
```

D.11.3.13 squareRoot

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
squareRoot(8315890421938608,32003);
↦ 18784
```

D.11.3.14 solutionsMod2

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
matrix M[3][3]=1,2,3,4,5,6,7,6,5;
solutionsMod2(M);
↦ 1,0,1
```

D.11.3.15 powerX

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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.11.3.16 babyGiant

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number b=2;
number y=10;
number p=101;
babyGiant(b,y,p);
↦ 25
```

D.11.3.17 rho

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,x,dp;
number b=2;
number y=10;
number p=101;
rho(b,y,p);
↦ 25
```

D.11.3.18 MillerRabin

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number x=2;
x=x^787-1;
MillerRabin(x,3);
↦ 0
```

D.11.3.19 SolowayStrassen

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number h=10;
number p=h^100+267;
//p=h^100+43723;
//p=h^200+632347;
SolowayStrassen(h,3);
↦ 0
```

D.11.3.20 PocklingtonLehmer

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number 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.11.3.21 PollardRho

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number h=10;
number p=h^30+4;
PollardRho(p,5000,0);
↦ [1]:
↦    2
↦ [2]:
↦    157
↦ [3]:
↦    18737561
↦ [4]:
↦    84982068258408294013
```

D.11.3.22 pFactor

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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 devides gcd(a^k-1,n) for some random a

Example:

```
LIB "crypto.lib";
ring R = 0,z,dp;
list L=primList(1000);
pFactor(1241143,13,L);
↦ 547
number h=10;
h=h^30+25;
pFactor(h,20,L);
↦ 325
```

D.11.3.23 quadraticSieve

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
list L=primList(5000);
quadraticSieve(7429,3,L,4);
↦ 17
quadraticSieve(1241143,100,L,50);
↦ 547
```

D.11.3.24 isOnCurve

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
isOnCurve(32003,5,7,list(10,16,1));
↦ 0
```

D.11.3.25 ellipticAdd

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number N=11;
number a=1;
number 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.11.3.26 ellipticMult

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number N=11;
number a=1;
number 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.11.3.27 ellipticRandomCurve

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
ellipticRandomCurve(32003);
↦ [1]:
↦    22857
```

```
↦ [2]:
↦    24963
↦ [3]:
↦    1
```

D.11.3.28 ellipticRandomPoint

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
ellipticRandomPoint(32003,3,181);
↦ [1]:
↦    22857
↦ [2]:
↦    17476
↦ [3]:
↦    1
```

D.11.3.29 countPoints

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
countPoints(181,71,150);
↦ 198
```

D.11.3.30 ellipticAllPoints

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
list L=ellipticAllPoints(181,71,150);
size(L);
↦ 198
L[size(L)];
↦ [1]:
```

```
↦     179
↦ [2]:
↦     0
↦ [3]:
↦     1
```

D.11.3.31 ShanksMestre

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
ShanksMestre(32003,71,602);
↦ 32021
```

D.11.3.32 Schoof

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
Schoof(32003,71,602);
↦ 32021
```

D.11.3.33 generateG

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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.11.3.34 factorLenstraECM

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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 dedected using the operations on the elliptic curve
- is similarly to Pollard's p-1-factorization

Example:

```
LIB "crypto.lib";
ring R = 0,z,dp;
list L=primList(1000);
factorLenstraECM(181*32003,L,10,5);
↦ 181
number h=10;
h=h^30+25;
factorLenstraECM(h,L,4,3);
↦ 13
```

D.11.3.35 ECPP

Procedure from library `crypto.lib` (see Section D.11.3 [crypto_lib], page 1152).

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";
ring R = 0,z,dp;
number N=1267985441;
ECPP(N);
↦ [1]:
↦    [1]:
↦       67394594
↦    [2]:
```

```
↦        380636642
↦ [2]:
↦    [1]:
↦       780306204
↦    [2]:
↦       1106324420
↦    [3]:
↦       1
↦ [3]:
↦    1267993236
↦ [4]:
↦    105666103
```

D.11.4 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.11.4.1 ishyper

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.2 isoncurve

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.3 chinrestp

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.4 norm

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.5 multi

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.6 divisor

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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 additonally returns a parameter, which says, whether irreducible polynomials occured 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]:
↦       1
↦    [3]:
↦       2
↦ [2]:
↦    [1]:
↦       1
↦    [2]:
↦       -2
↦    [3]:
↦       2
```

```
↦ [3]:
↦    [1]:
↦       -1
↦    [2]:
↦       -3
↦    [3]:
↦       2
↦ 0
```

D.11.4.7 gcddivisor

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.8 semidiv

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.9 cantoradd

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.10 cantorred

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.11 double

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.4.12 cantormult

Procedure from library `hyperel.lib` (see Section D.11.4 [hyperel_lib], page 1165).

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.11.5 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.11.5.1 ecart

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.2 tail

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.3 sameComponent

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.4 leadmonomial

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.5 monomialLcm

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.6 spoly

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.7 minEcart

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.8 NFMora

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.9 prodcrit

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.10 chaincrit

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.11 pairset

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.12 updatePairs

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.13 standard

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.5.14 localstd

Procedure from library `teachstd.lib` (see Section D.11.5 [teachstd_lib], page 1171).

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.11.6 weierstr_lib

Library: weierstr.lib

Purpose: Procedures for the Weierstrass Theorems

Author: G.-M. Greuel, greuel@mathematik.uni-kl.de

Procedures:

D.11.6.1 weierstrDiv

Procedure from library `weierstr.lib` (see Section D.11.6 [weierstr_lib], page 1178).

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.11.6.2 weierstrPrep

Procedure from library `weierstr.lib` (see Section D.11.6 [weierstr_lib], page 1178).

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.11.6.3 lastvarGeneral

Procedure from library `weierstr.lib` (see Section D.11.6 [weierstr_lib], page 1178).

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.11.6.4 generalOrder

Procedure from library `weierstr.lib` (see Section D.11.6 [weierstr_lib], page 1178).

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.11.7 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.11.7.1 nrRootsProbab

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.2 [nrRootsDeterm], page 1182; Section D.11.8.15 [nrroots], page 1195; Section D.11.7.9 [randcharpoly], page 1185; Section D.7.2.2 [solve], page 1034.

D.11.7.2 nrRootsDeterm

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.1 [nrRootsProbab], page 1181; Section D.11.8.15 [nrroots], page 1195; Section D.7.2.2 [solve], page 1034; Section D.11.7.4 [sturmquery], page 1182.

D.11.7.3 symsignature

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.5 [matbil], page 1183; Section D.11.7.4 [sturmquery], page 1182.

D.11.7.4 sturmquery

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.5 [matbil], page 1183; Section D.11.7.3 [symsignature], page 1182.

D.11.7.5 matbil

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.6 [matmult], page 1183; Section D.11.7.7 [tracemult], page 1184.

D.11.7.6 matmult

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.8 [coords], page 1184; Section D.11.7.5 [matbil], page 1183.

D.11.7.7 tracemult

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.6 [matmult], page 1183; Section 5.1.139 [trace], page 230.

D.11.7.8 coords

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.5 [matbil], page 1183; Section D.11.7.6 [matmult], page 1183.

D.11.7.9 randcharpoly

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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+1720335326522z6-84457573876829350 8z5+257398574985069575 97394\
   5z4-498553962538427861265995664 42z3+599150644929810240784558288657617 2z2-\
   40833586518340765147334336216299817 7144z+120782617595757843238663349007 81\
   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*z6-267192014719731278 0*z5+1191863249059288760\
   005489*z4-337242235263204293461543939056*z3+590799520413827288089564257 46\
   100736*z2-585536730347248462296397514395385856000 0*z+251206293137619500 33\
   639593091861053440000000 0
↦ ************************************************************************
↦ * 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.11.7.10 verify

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.9 [randcharpoly], page 1185.

D.11.7.11 randlinpoly

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.9 [randcharpoly], page 1185.

D.11.7.12 powersums

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.13 [symmfunc], page 1187.

D.11.7.13 symmfunc

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.12 [powersums], page 1187.

D.11.7.14 univarpoly

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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.11.7.15 qbase

Procedure from library `rootsmr.lib` (see Section D.11.7 [rootsmr_lib], page 1181).

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 5.1.60 [kbase], page 168.

D.11.8 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.11.8.1 isuni

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.2 [whichvariable], page 1190.

D.11.8.2 whichvariable

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.1 [isuni], page 1189.

D.11.8.3 varsigns

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.5 [boundposDes], page 1191.

D.11.8.4 boundBuFou

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.5 [boundposDes], page 1191; Section D.11.8.3 [varsigns], page 1190.

D.11.8.5 boundposDes

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.4 [boundBuFou], page 1190.

D.11.8.6 boundDes

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.4 [boundBuFou], page 1190.

D.11.8.7 allrealst

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.9 [allreal], page 1192; Section D.11.8.10 [sturm], page 1193; Section D.11.8.12 [sturmha], page 1194.

D.11.8.8 maxabs

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.10 [sturm], page 1193.

D.11.8.9 allreal

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.7 [allrealst], page 1192.

D.11.8.10 sturm

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.9 [allreal], page 1192; Section D.11.8.7 [allrealst], page 1192; Section D.11.8.12 [sturmha], page 1194.

D.11.8.11 sturmseq

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.10 [sturm], page 1193; Section D.11.8.13 [sturmhaseq], page 1194.

D.11.8.12 sturmha

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.9 [allreal], page 1192; Section D.11.8.10 [sturm], page 1193.

D.11.8.13 sturmhaseq

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.10 [sturm], page 1193; Section D.11.8.12 [sturmha], page 1194; Section D.11.8.11 [sturmseq], page 1193.

D.11.8.14 reverse

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.15 nrroots

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.11.8.5 [boundposDes], page 1191; Section D.11.8.10 [sturm], page 1193; Section D.11.8.12 [sturmha], page 1194.

D.11.8.16 isparam

Procedure from library `rootsur.lib` (see Section D.11.8 [rootsur_lib], page 1189).

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.12 Tropical Geometry

D.12.1 polymake_lib

Library: polymake.lib

Purpose: Computations with polytopes and fans, interface to polymake and TOPCOM

Author: Thomas Markwig, email: keilen@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 the following note or consult the help string of the procedures.

Note: Even though this is a @sc{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 @sc{Singular} but by the program
- polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, TU Darmstadt (see http://www.math.tu-berlin.de/polymake/),
respectively (only in the procedure triangularions) by the program
- topcom by Joerg Rambau, Universitaet Bayreuth (see http://www.uni-bayreuth.de/departments/wirtschaftsmathematik/rambau/TOPCOM);
This library should rather be seen as an interface which allows to use a (very limited) number of options which polymake respectively topcom offers to compute with polytopes and fans and to make the results available in @sc{Singular} for further computations;
moreover, the user familiar with @sc{Singular} does not have to learn the syntax of polymake or topcom, if the options offered here are sufficient for his purposes.
Note, though, that the procedures concerned with planar polygons are independent of both, polymake and topcom.

Procedures using polymake: Procedures using topcom: Procedures using polymake and topcom: Procedures concerned with planar polygons: Auxilary procedures:

D.12.1.1 polymakePolytope

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: polymakePolytope(polytope[,#]); polytope list, # string

Assume: each row of polytope gives the coordinates of a lattice point of a polytope with their affine coordinates as given by the output of secondaryPolytope

Purpose: the procedure calls polymake to compute the vertices of the polytope as well as its dimension and information on its facets

Return: list L with four entries
L[1] : an integer matrix whose rows are the coordinates of vertices of the polytope
L[2] : the dimension of the polytope
L[3] : a list whose i-th entry explains to which vertices the ith vertex of the Newton polytope is connected
– i.e. L[3][i] is an integer vector and an entry k in there means that the vertex L[1][i] is connected to the vertex L[1][k]
L[4] : an integer matrix whose rows mulitplied by (1,var(1),...,var(nvar)) give a linear system of equations describing the affine hull of the polytope,
i.e. the smallest affine space containing the polytope

Note: - for its computations the procedure calls the program polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, TU Darmstadt; it therefore is necessary that this program is installed in order to use this procedure;
see http://www.math.tu-berlin.de/polymake/
- note that in the vertex edge graph we have changed the polymake convention which starts indexing its vertices by zero while we start with one !
- the procedure creates the file /tmp/polytope.polymake which contains the polytope in polymake format; if you wish to use this for further computations with polymake, you have to use the procedure polymakeKeepTmpFiles in before
- moreover, the procedure creates the file /tmp/polytope.output which it deletes again before ending
- it is possible to provide an optional second argument a string which then will be used instead of 'polytope' in the name of the polymake output file

Example:

```
LIB "polymake.lib";
// the lattice points of the unit square in the plane
list points=intvec(0,0),intvec(0,1),intvec(1,0),intvec(1,1);
// the secondary polytope of this lattice point configuration is computed
intmat secpoly=secondaryPolytope(points)[1];
list np=polymakePolytope(secpoly);
// the vertices of the secondary polytope are:
np[1];
// its dimension is
np[2];
// np[3] contains information how the vertices are connected to each other,
// e.g. the first vertex (number 0) is connected to the second one
np[3][1];
// the affine hull has the equation
ring r=0,x(1..4),dp;
matrix M[5][1]=1,x(1),x(2),x(3),x(4);
np[4]*M;
```

D.12.1.2 newtonPolytope

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: newtonPolytope(f[,#]); f poly, # string

Return: list L with four entries
L[1] : an integer matrix whose rows are the coordinates of vertices of the Newton polytope of f
L[2] : the dimension of the Newton polytope of f
L[3] : a list whose ith entry explains to which vertices the ith vertex of the Newton polytope is connected
– i.e. L[3][i] is an integer vector and an entry k in there means that the vertex L[1][i] is
connected to the vertex L[1][k]
L[4] : an integer matrix whose rows mulitplied by (1,var(1),...,var(nvar)) give a linear system of equations describing the affine hull of the Newton polytope, i.e. the smallest affine space containing the Newton polytope

Note: - if we replace the first column of L[4] by zeros, i.e. if we move the affine hull to the origin, then we get the equations for the orthogonal complement of the linearity space of the normal fan dual to the Newton polytope, i.e. we get the EQUATIONS that we need as input for polymake when computing the normal fan
- the procedure calls for its computation polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, so it only works if polymake is installed; see http://www.math.tu-berlin.de/polymake/
- the procedure creates the file /tmp/newtonPolytope.polymake which contains the polytope in polymake format and which can be used for further computations with polymake
- moreover, the procedure creates the file /tmp/newtonPolytope.output and deletes it again before ending
- it is possible to give as an optional second argument a string which then will be used instead of 'newtonPolytope' in the name of the polymake output file

Example:

```
LIB "polymake.lib";
ring r=0,(x,y,z),dp;
matrix M[4][1]=1,x,y,z;
poly f=y3+x2+xy+2xz+yz+z2+1;
// the Newton polytope of f is
list np=newtonPolytope(f);
// the vertices of the Newton polytope are:
np[1];
// its dimension is
np[2];
// np[3] contains information how the vertices are connected to each other,
// e.g. the first vertex (number 0) is connected to the second, third and
//      fourth vertex
np[3][1];
//////////////////////////
f=x2-y3;
// the Newton polytope of f is
np=newtonPolytope(f);
// the vertices of the Newton polytope are:
```

```
np[1];
// its dimension is
np[2];
// the Newton polytope is contained in the affine space given
//     by the equations
np[4]*M;
```

D.12.1.3 newtonPolytopeLP

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: newtonPolytopeLP(f); f poly

Return: list, the exponent vectors of the monomials occuring in f, i.e. the lattice points of the Newton polytope of f

Example:

```
LIB "polymake.lib";
ring r=0,(x,y,z),dp;
poly f=y3+x2+xy+2xz+yz+z2+1;
// the lattice points of the Newton polytope of f are
newtonPolytopeLP(f);
```

D.12.1.4 normalFan

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: normalFan (vert,aff,graph,rays,[,#]); vert,aff intmat, graph list, rays int, # string

Assume: - vert is an integer matrix whose rows are the coordinate of the vertices of a convex lattice polytope;
- aff describes the affine hull of this polytope, i.e. the smallest affine space containing it, in the following sense: denote by n the number of columns of vert, then multiply aff by (1,x(1),...,x(n)) and set the resulting terms to zero in order to get the equations for the affine hull;
- the ith entry of graph is an integer vector describing to which vertices the ith vertex is connected, i.e. a k as entry means that the vertex vert[i] is connected to vert[k];
- the integer rays is either one (if the extreme rays should be computed) or zero (otherwise)

Return: list, the ith entry of L[1] contains information about the cone in the normal fan dual to the ith vertex of the polytope
L[1][i][1] = integer matrix representing the inequalities which describe the cone dual to the ith vertex
L[1][i][2] = a list which contains the inequalities represented by L[i][1] as a list of strings, where we use the
variables x(1),...,x(n)
L[1][i][3] = only present if 'er' is set to 1; in that case it is an interger matrix whose rows are the extreme rays
of the cone
L[2] = is an integer matrix whose rows span the linearity space of the fan, i.e. the linear space which is contained in each cone

Note: - the procedure calls for its computation polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, so it only works if polymake is installed;

see http://www.math.tu-berlin.de/polymake/
- in the optional argument # it is possible to hand over other names for the variables to be used – be careful, the format must be correct which is not tested, e.g. if you want the variable names to be u00,u10,u01,u11 then you must hand over the string 'u11,u10,u01,u11'

Example:

```
LIB "polymake.lib";
ring r=0,(x,y,z),dp;
matrix M[4][1]=1,x,y,z;
poly f=y3+x2+xy+2xz+yz+z2+1;
// the Newton polytope of f is
list np=newtonPolytope(f);
// the Groebner fan of f, i.e. the normal fan of the Newton polytope
list gf=normalFan(np[1],np[4],np[3],1,"x,y,z");
// the number of cones in the Groebner fan of f is:
size(gf[1]);
// the inequalities of the first cone as matrix are:
print(gf[1][1][1]);
// the inequalities of the first cone as string are:
print(gf[1][1][2]);
// the rows of the following matrix are the extreme rays of the first cone:
print(gf[1][1][3]);
// each cone contains the linearity space spanned by:
print(gf[2]);
```

D.12.1.5 groebnerFan

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: groebnerFan(f[,#]); f poly, # string

Return: list, the ith entry of L[1] contains information about the ith cone in the Groebner fan dual to the ith vertex in the Newton polytope of the f
L[1][i][1] = integer matrix representing the inequalities which describe the cone
L[1][i][2] = a list which contains the inequalities represented by L[1][i][1] as a list of strings
L[1][i][3] = an interger matrix whose rows are the extreme rays of the cone
L[2] = is an integer matrix whose rows span the linearity space of the fan, i.e. the linear space which is contained
in each cone
L[3] = the Newton polytope of f in the format of the procedure newtonPolytope
L[4] = integer matrix where each row represents the exponet vector of one monomial occuring in the input polynomial

Note: - if you have already computed the Newton polytope of f then you might want to use the procedure normalFan instead in order to avoid doing costly computation twice
- the procedure calls for its computation polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, so it only works if polymake is installed; see http://www.math.tu-berlin.de/polymake/
- the procedure creates the file /tmp/newtonPolytope.polymake which contains the Newton polytope of f in polymake format and which can be used for further computations with polymake

- it is possible to give as an optional second argument as string which then will be used instead of 'newtonPolytope' in the name of the polymake output file

Example:

```
LIB "polymake.lib";
ring r=0,(x,y,z),dp;
matrix M[4][1]=1,x,y,z;
poly f=y3+x2+xy+2xz+yz+z2+1;
// the Newton polytope of f is
list gf=groebnerFan(f);
// the exponent vectors of f are ordered as follows
gf[4];
// the first cone of the groebner fan has the inequalities
gf[1][1][1];
// as a string they look like
gf[1][1][2];
// and it has the extreme rays
print(gf[1][1][3]);
// the linearity space is spanned by
print(gf[2]);
// the vertices of the Newton polytope are:
gf[3][1];
// its dimension is
gf[3][2];
// np[3] contains information how the vertices are connected to each other,
// e.g. the 1st vertex is connected to the 2nd, 3rd and 4th vertex
gf[3][3][1];
```

D.12.1.6 intmatToPolymake

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: intmatToPolymake(M,art); M intmat, art string

Assume: - M is an integer matrix which should be transformed into polymake format;
- art is one of the following strings:
+ 'rays' : indicating that a first column of 0's should be added
+ 'points' : indicating that a first column of 1's should be added

Return: string, the matrix is transformed in a string and a first column has been added

Example:

```
LIB "polymake.lib";
intmat M[3][4]=1,2,3,4,5,6,7,8,9,10,11,12;
intmatToPolymake(M,"rays");
intmatToPolymake(M,"points");
```

D.12.1.7 polymakeToIntmat

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: polymakeToIntmat(pm,art); pm, art string

Assume: pm is the result of calling polymake with one 'argument' like VERTICES, AFFINE_HULL, etc., so that the first row of the string is the name of the corresponding 'argument' and the further rows contain the result which consists of vectors either over the integers or over the rationals

Return: intmat, the rows of the matrix are basically the vectors in pm, starting from the second row, where each row has been multiplied with the lowest common multiple of the denominators of its entries as if it is an integer matrix; moreover, if art==’affine’, then the first column is omitted since we only want affine coordinates

Example:

```
LIB "polymake.lib";
// this is the usual output of some polymake computation
string pm="VERTICES
0 1 3 5/3 1/3 -1 -23/3 -1/3 5/3 1/3 1
0 1 3 -23/3 5/3 1 5/3 1/3 1/3 -1/3 -1
0 1 1 1/3 -1/3 -1 5/3 1/3 -23/3 5/3 3
0 1 1 5/3 -23/3 3 1/3 5/3 -1/3 1/3 -1
0 1 -1 1/3 5/3 3 -1/3 -23/3 1/3 5/3 1
0 1 -1 -1/3 1/3 1 1/3 5/3 5/3 -23/3 3
0 1 -1 1 3 -5 -1 3 -1 1 -1
0 1 -1 -1 -1 -1 1 1 3 3 -5
0 1 -5 3 1 -1 3 -1 1 -1 -1
";
intmat PM=polymakeToIntmat(pm,"affine");
// note that the first column has been removed, since we asked for
// affine coordinates, and the denominators have been cleared
print(PM);
```

D.12.1.8 triangulations

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: triangulations(polygon); list polygon

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.uni-bayreuth.de/departments/wirtschaftsmathematik/rambau/TOPCOM

- 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 use the procedure polymakeKeepTmpFiles in before
- 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);
triang;
```

D.12.1.9 secondaryPolytope

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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.uni-bayreuth.de/departments/wirtschaftsmathematik/rambau/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);
// the points in the secondary polytope
print(secpoly[1]);
// the corresponding triangulations
secpoly[2];
```

D.12.1.10 secondaryFan

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: secondaryFan(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, the ith entry of L[1] contains information about the ith cone in the secondary fan of the polygon, i.e. the cone dual to the ith vertex of the secondary polytope
L[1][i][1] = integer matrix representing the inequalities which describe the cone dual to the ith vertex
L[1][i][2] = a list which contains the inequalities represented by L[1][i][1] as a list of strings, where we use the
variables x(1),...,x(n)
L[1][i][3] = only present if 'er' is set to 1; in that case it is an interger matrix whose rows are the extreme rays
of the cone
L[2] = is an integer matrix whose rows span the linearity space of the fan, i.e. the linear space which is contained in each cone
L[3] = the secondary polytope in the format of the procedure polymakePolytope
L[4] = the list of triangulations corresponding to the vertices of the secondary polytope

Note: - the procedure calls for its computation polymake by Ewgenij Gawrilow, TU Berlin and Michael Joswig, so it only works if polymake is installed; see http://www.math.tu-berlin.de/polymake/
- in the optional argument # it is possible to hand over other names for the variables to be used – be careful, the format must be correct which is not tested, e.g. if you want the variable names to be u00,u10,u01,u11 then you must hand over the string 'u11,u10,u01,u11'
- 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.uni-bayreuth.de/departments/wirtschaftsmathematik/rambau/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 secfan=secondaryFan(polygon);
// the number of cones in the secondary fan of the polygon
size(secfan[1]);
// the inequalities of the first cone as matrix are:
print(secfan[1][1][1]);
// the inequalities of the first cone as string are:
print(secfan[1][1][2]);
// the rows of the following matrix are the extreme rays of the first cone:
print(secfan[1][1][3]);
// each cone contains the linearity space spanned by:
print(secfan[2]);
// the points in the secondary polytope
```

```
print(secfan[3][1]);
// the corresponding triangulations
secfan[4];
```

D.12.1.11 cycleLength

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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);
```

D.12.1.12 splitPolygon

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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];
// the points on facets which are not vertices
sp[2];
// the interior points
sp[3];
```

D.12.1.13 eta

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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 from the vertices with indices i, j and k

Return: intvec, the integer vector eta describing that vertex of the Newton polytope discriminant of the polygone 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);
```

D.12.1.14 findOrientedBoundary

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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];
// the vertices of the boundary are boundarypolygon[2]
boundarypolygon[2];
```

D.12.1.15 cyclePoints

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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);
```

D.12.1.16 latticeArea

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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);
```

D.12.1.17 picksFormula

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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];
// the number of lattice points on the boundary is:
pick[2];
// the number of interior lattice points is:
```

```
pick[3];
// the number's are related by Pick's formula:
pick[1]-pick[2]-2*pick[3]+2;
```

D.12.1.18 ellipticNF

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

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=newtonPolytopeLP(f);
// its lattice points are
polygon;
// find its normal form
list nf=ellipticNF(polygon);
// the vertices of the normal form are
nf[1];
// it has been transformed by the unimodular affine transformation A*x+v
// with matrix A
nf[2];
// and translation vector v
nf[3];
// the 3rd lattice point ...
polygon[3];
// ... has been transformed to
nf[4][3];
```

D.12.1.19 ellipticNFDB

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: ellipticNFDB(n[,#]); n int, # list

Assume: n is an integer between 1 and 16

Purpose: this is a database storing the 16 normal forms of planar polygons with precisely one interior point up to unimodular affine transformations
(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 nth normal form
L[2] : list whose entries are all the lattice points of the nth normal form
L[3] : only present if the optional parameter # is present, and then it is a polynomial in the variables (x,y) whose
Newton polygon is the nth normal form

Note: the optional parameter is only allowed if the basering has the variables x and y

Example:

```
LIB "polymake.lib";
list nf=ellipticNFDB(5);
// the vertices of the 5th normal form are
nf[1];
// its lattice points are
nf[2];
```

D.12.1.20 polymakeKeepTmpFiles

Procedure from library `polymake.lib` (see Section D.12.1 [polymake_lib], page 1196).

Usage: polymakeKeepTmpFiles(int i); i int

Purpose: some procedures create files in the directory /tmp which are used for computations with polymake respectively topcom; these will be removed when the corresponding procedure is left; however, it might be desireable to keep them for further computations with either polymake or topcom; this can be achieved by this procedure; call the procedure as:
- polymakeKeepTmpFiles(1); - then the files will be kept
- polymakeKeepTmpFiles(0); - then files will be removed in the future

Return: none

D.12.2 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

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 kghostview 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 @sc{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, whereever 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

- 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
- jInvariant computes the j-invariant of an elliptic curve
- weierstrassForm computes the Weierstrass form of an elliptic curve

Procedures for tropical lifting: Procedures for drawing tropical curves: General procedures: Procedures for latex conversion: Auxilary procedures:

D.12.2.1 tropicalLifting

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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
'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 OPITON '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 @sc{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 @sc{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";
//////////////////////////////////////////////////////
```

```
// 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);
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;
// Since the computations were done over Q a field extension was necessary,
// and the parameter "a" satisfies the equation given by minpoly
minpoly;
////////////////////////////////////////////////////
// 2nd EXAMPLE
////////////////////////////////////////////////////
setring r;
LIST=tropicalLifting(x12-t11,intvec(12,-11),2,"isPrime","isInTrop");
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;
LIST[3];
////////////////////////////////////////////////////
// 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);
// 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;
LIST[4];
// An easier way to display this is via displayTropicalLifting.
setring R;
displayTropicalLifting(LIST,"subst");
```

D.12.2.2 displayTropicalLifting

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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");
```

D.12.2.3 tropicalCurve

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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 monmials 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;
// the coordinates of the first vertex are graph[1][1],graph[1][2];
graph[1][1],graph[1][2];
// 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])];
// 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])];
// from the first vertex emerge size(graph[1][4]) unbounded edges
size(graph[1][4]);
// the primitive integral direction vector of the first unbounded edge
//     of the first vertex
graph[1][4][1][1];
// the weight of the first unbounded edge of the first vertex
graph[1][4][1][2];
// the monomials which are part of the Newton subdivision of the first vertex
graph[1][5];
// connecting the points in graph[size(graph)][1] we get
//     the boundary of the Newton polytope
graph[size(graph)][1];
// 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];
```

D.12.2.4 drawTropicalCurve

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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 kghostview; 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 kghostview 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 kghostview
drawTropicalCurve(f);
// 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;
drawTropicalCurve(tropical_g,"max");
```

D.12.2.5 drawNewtonSubdivision

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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 kghostview; 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 kghostview
drawNewtonSubdivision(f);
// 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;
drawNewtonSubdivision(tropical_g);
```

D.12.2.6 tropicalJInvariant

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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 weigthed 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);
// the Newton polygone need not be the standard simplex
tropicalJInvariant(x+y+x2y+xy2+1/t*xy);
// the curve can have arbitrary degree
tropicalJInvariant(t*(x7+y7+1)+1/t*(x4+y4+x2+y2+x3y+xy3)+1/t7*x2y2);
// 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);
// but it does realise, if the curve has no loop at all ...
tropicalJInvariant(x+y+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);
```

D.12.2.7 weierstrassForm

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

Usage: weierstrassForm(wf[,#]); wf poly, # list

Assume: wf is a a polynomial whose Newton polygon has precisely one interior lattice point, so that it defines an elliptic curve on the toric surface corresponding to the Newton polygon

Return: poly, the Weierstrass normal form of the polynomial

Note: - the algorithm for the coefficients of the Weierstrass form is due to Fernando Rodriguez Villegas, villegas@math.utexas.edu
- the characteristic of the base field should not be 2 or 3
- if an additional argument # is given, a simplified Weierstrass form is computed

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),lp;
// f is already in Weierstrass form
poly f=y2+yx+3y-x3-2x2-4x-6;
weierstrassForm(f);
// g is not, but wg is
poly g=x+y+x2y+xy2+1/t*xy;
poly wg=weierstrassForm(g);
wg;
// but it is not yet simple, since it still has an xy-term, unlike swg
poly swg=weierstrassForm(g,1);
swg;
// the j-invariants of all three polynomials coincide
jInvariant(g);
jInvariant(wg);
jInvariant(swg);
// the following curve is elliptic as well
poly h=x22y11+x19y10+x17y9+x16y9+x12y7+x9y6+x7y5+x2y3;
// its Weierstrass form is
weierstrassForm(h);
```

D.12.2.8 jInvariant

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

Usage: jInvariant(f[,#]); f poly, # list

Assume: - f is a a polynomial whose Newton polygon has precisely one interior lattice point, so that it defines an elliptic curve on the toric surface corresponding to the Newton polygon
- it the optional argument # is present the base field should be Q(t) and the optional argument should be one of the following strings:
'ord' : then the return value is of type integer, namely the order of the j-invariant
'split' : then the return value is a list of two polynomials, such that the quotient of these two is the j-invariant

Return: poly, the j-invariant of the elliptic curve defined by poly

Note: the characteristic of the base field should not be 2 or 3, unless the input is a plane cubic

Example:

```
LIB "tropical.lib";
ring r=(0,t),(x,y),dp;
```

```
// jInvariant computes the j-invariant of a cubic
jInvariant(x+y+x2y+y3+1/t*xy);
// if the ground field has one parameter t, then we can instead
//    compute the order of the j-invariant
jInvariant(x+y+x2y+y3+1/t*xy,"ord");
// one can compare the order of the j-invariant to the tropical j-invariant
tropicalJInvariant(x+y+x2y+y3+1/t*xy);
// the following curve is elliptic as well
poly h=x22y11+x19y10+x17y9+x16y9+x12y7+x9y6+x7y5+x2y3+x14y8;
// its j-invariant is
jInvariant(h);
```

D.12.2.9 conicWithTangents

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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];
// conic[2] is the equation of the conic f passing through the five points
conic[2];
// conic[3] is a list containing the equations of the tangents
//          through the five points
conic[3];
// conic[4] is an ideal representing the tropicalisation of the conic f
conic[4];
// conic[5] is a list containing the tropicalisation
//          of the five tangents in conic[3]
conic[5];
// conic[6] is a list containing the vertices of the tropical conic
conic[6];
// conic[7] is a list containing the vertices of the five tangents
```

```
conic[7];
// 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 kghostview
write(":w /tmp/conic.tex",conic[8]);
system("sh","cd /tmp; latex /tmp/conic.tex; dvips /tmp/conic.dvi -o;
kghostview conic.ps &");
// 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
```

D.12.2.10 tropicalise

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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));
```

D.12.2.11 tropicaliseSet

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.12 tInitialForm

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.13 tInitialIdeal

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
// and they do not generate the t-initial ideal of i
tInitialIdeal(i,w);
```

D.12.2.14 initialForm

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.15 initialIdeal

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.16 texNumber

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.17 texPolynomial

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.18 texMatrix

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.19 texDrawBasic

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.20 texDrawTropical

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
// compute the texdraw code again, but set the scalefactor to 1
texDrawTropical(graph,"",1);
```

D.12.2.21 texDrawNewtonSubdivision

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.22 texDrawTriangulation

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.23 radicalMemberShip

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
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);
```

D.12.2.24 tInitialFormPar

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.25 tInitialFormParMax

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.26 solveTInitialFormPar

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.27 detropicalise

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.28 tDetropicalise

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.29 dualConic

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
```

D.12.2.30 parameterSubstitute

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
parameterSubstitute(f,-1);
```

D.12.2.31 tropicalSubst

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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);
// 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;
// - the j-invariant, and hence its valuation,
//   does not change under the transformation
jInvariant(f,"ord");
// - 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);
// - b=1, then it is still true, but only just ...
list g1=tropicalSubst(f,1,x,x+t,y,y);
tropicalJInvariant(g1);
// - 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);
```

D.12.2.32 randomPoly

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

Usage: randomPoly(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;
randomPoly(3,-2,5);
randomPoly(3,-2,5,1);
```

D.12.2.33 cleanTmp

Procedure from library `tropical.lib` (see Section D.12.2 [tropical_lib], page 1210).

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 Contributed

D.13.1 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.13.1.1 signatureL

Procedure from library `phindex.lib` (see Section D.13.1 [phindex_lib], page 1229).

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.13.1.2 signatureLqf

Procedure from library `phindex.lib` (see Section D.13.1 [phindex_lib], page 1229).

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.13.1.3 PH_ais

Procedure from library `phindex.lib` (see Section D.13.1 [phindex_lib], page 1229).

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.13.1.4 PH_nais

Procedure from library `phindex.lib` (see Section D.13.1 [phindex_lib], page 1229).

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
```

8 Release Notes

8.1 News and changes

NEWS in SINGULAR 3-1-1

News for version 3-1-1

- new option `qringNF`, see Section 5.1.98 [option], page 192.
- new system command `system("cpu")`, see Section 5.1.137 [system], page 227.

New SINGULAR functions

- new command: `farey`: lifting to Q (see Section 5.1.31 [farey], page 148)
- new command: `monomial` (see Section 5.1.85 [monomial], page 186)
- command extended: `liftstd` also computes syzygies. (see Section 5.1.72 [liftstd], page 175)
- command extended: `minor` has more options. (see Section 5.1.81 [minor], page 182)
- command extended: `opposite` (see Section 7.3.20 [opposite], page 297)

Internal Changes

- new minor code
- removed `EXTGCD` (use `extgcd`)
- moved mp_set_memory_functions-call from kernel/mminit.cc to tesths.cc:main (in order not to call it for libsingular)

New SINGULAR libraries

- new library: normaliz.lib (see Section D.4.16 [normaliz_lib], page 760: Interface to Normaliz 2.2)

Changed SINGULAR libraries

homolog.lib (Section D.4.7 [homolog_lib], page 684): canonMap

dmod.lib (Section 7.7.3 [dmod_lib], page 346): operatorModulo

News for version 3-1-0

- new coefficients: Z, Z/m, Z/(2^n) (see Section 3.3 [Rings and orderings], page 29)
- new handling of the default argument in libraries (see Section 3.7.3 [Parameter list], page 51)
- `ESingular` updated for emacs 22
- licences for all parts of SINGULAR clarified (see Chapter 1 [Preface], page 1)

New SINGULAR functions

- new command: kernel (see Section 5.1.61 [kernel], page 169)
- new command: sqrfree (see Section 5.1.129 [sqrfree], page 219)
- command changed: the first argument to `monitor` should be an ASCII link. (see Section 5.1.84 [monitor], page 185)
- command extended: eliminate: variables to eliminate may also be given as intvec. (see Section 5.1.23 [eliminate], page 143)

Internal Changes

- handling of large input for std improved
- Section 5.1.55 [interred], page 165 implemented in a different way
- Section 5.1.60 [kbase], page 168 honors the attribute "isHomog"
- Section 5.1.57 [jacob], page 166 accepts modules and matrices
- Section 5.1.41 [gcd], page 154 over algebraic extensions of the rationals implemented in a different way
- new build target: libsingular.a (for gfan etc.)
- code variants now depend on CPU type, not OS
- better test for built-in limits (see Section 6.1 [Limitations], page 254)
- operator `new(size_t,const std::nothrow_t&)` now also overloaded

New SINGULAR libraries

- surfex: new version 0.90 (see Section D.8.5 [surfex_lib], page 1072).
- new library: redcgs.lib (see Section D.2.7 [redcgs_lib], page 567: Reduced Comprehensive Groebner Systems)
- new library: tropical.lib (see Section D.12.2 [tropical_lib], page 1210: Computations in Tropical Geometry)
- new library: polymake.lib (see Section D.12.1 [polymake_lib], page 1196: Computations with polytopes and fans, interface to polymake and TOPCOM)
- new library: sing4ti2.lib (see Section D.4.27 [sing4ti2_lib], page 875: interface to 4ti2
- new library: decodegb.lib (see Section D.9.2 [decodegb_lib], page 1090: Generating and solving systems of polynomial equations for decoding and finding the minimum distance of linear codes)
- new library: dmodapp.lib (see Section 7.7.4 [dmodapp_lib], page 364: applications of D-modules)
- new library: bfun.lib global (see Section 7.7.1 [bfun_lib], page 321: Bernstein-Sato polynomial)
- new library: freegb.lib (see Section 7.7.5 [freegb_lib], page 375): Twosided Non-commutative Groebner bases in Free Algebras
- new library: jacobson.lib (see Section D.10.3 [jacobson_lib], page 1130): Algorithms for Smith and Jacobson Normal Form
- contributed library: cimonom.lib (see Section D.4.4 [cimonom_lib], page 671): determines if the toric ideal of an affine monomial curve is a complete intersection
- contributed library: phindex.lib (see Section D.13.1 [phindex_lib], page 1229): Poincare-Hopf index of a real analytic vector field

Changed SINGULAR libraries

normal.lib (Section D.4.15 [normal_lib], page 740): changed structure of the result, new algorithms have been implemented which improve the performance

elim.lib (Section D.4.5 [elim_lib], page 672): Section D.4.5.3 [elim], page 676, Section D.4.5.6 [nselect], page 678, Section D.4.5.8 [select], page 680, Section D.4.5.9 [select1], page 680: changed syntax

homolog.lib: kernel renamed to Section D.4.7.15 [hom_kernel], page 703.
(See also Section 5.1.61 [kernel], page 169, Section D.4.2.5 [alg_kernel], page 657).

matrix.lib (Section D.3.1 [matrix_lib], page 614): new commands for computing symmetric/exterior powers/bases

surf.lib: new command `surfer`: interface to program `surfer`
(See Section D.8.4 [surf_lib], page 1070).

teachstd.lib (Section D.11.5 [teachstd_lib], page 1171): spoly works now in non-commutative algebras and Section D.11.5.13 [standard], page 1177 can thus be used there. However, since product criterion is a priori not applicable in the non-commutative case, one may want to disable it first (see `prodcrit` for details).

many changes of names in libraries (to have a more consistent naming scheme)

News for version 3-0-4

- licence changed: due to the need to use stuff under (L)GPL3, all parts of SINGULAR are licenced under GPL (resp. LGPL) version 2, or (at your option) version 3
- new command: (Section 5.1.143 [univariate], page 232): test polynomials for being univariate.
- new command: (Section 5.1.147 [variables], page 233): ideal of all variables occurring in a polynomial, ideal or matrix.
- syntax change: Section 7.3.17 [ncalgebra], page 295 should be substituted by Section 7.3.16 [nc_algebra], page 292. `ncalgebra` is now depreciated, but still supported
- library Section 7.7.10 [nctools_lib], page 412 updated to use `nc_algebra`,
  Section 7.7.10.4 [Weyl], page 415, Section 7.7.10.7 [Exterior], page 417, Section 7.7.10.8 [findimAlgebra], page 417 do not change the current ring but return the new structure
- kernel: use Conway polynomials and support more finite fields

News in version 3-0-3

The version 3-0-3 is stabilizing release, a result of a long beta test and the integration of a lot of small fixes which were on our waiting list for integration.

It contains also a lot of new features:

- licence changed: omalloc and MP are now (also) available under GPL; that means that all parts of SINGULAR are licenced under GPL (resp. LGPL).
- factory, libfac, SINGULAR updated for gcc 4.1.x
- kernel updated for the optional use of boost.
- can now be built as a library.
- new operator `a:b` gives an `intvec` of length `b` with constant entries `a`
- new command: (Section 5.1.7 [chinrem], page 132): lifting via chinese remainder theorem
- new command: (Section 5.1.54 [interpolation], page 164): ideal of points with given multiplicities

- non-commutative kernel subsystem was rewritten in order to support specific algebras more efficiently. Implemented algebras at the moment: super-commutative algebras (in particular exterior algebras).
- Section 5.1.133 [std], page 223 et al.: new selection strategy for reductions (Section 5.1.98 [option], page 192 (length)).
- Section 5.1.115 [reduce], page 206: new strategy for selection and normalization.
- Section 5.1.125 [simplify], page 216 slightly changed: does not omit zero polynomial unless specified.
- new library: compregb.lib (Section D.2.2 [compregb_lib], page 528): comprehensive Groebner base system
- new library: kskernel.lib (Section D.5.10 [kskernel_lib], page 952): kernel of the kodaira-spencer map for irreducible plane curve singularities
- new library: modstd.lib (Section D.4.10 [modstd_lib], page 710): Groebner base computations over the rational numbers via modular computations
- new library: noether.lib (Section D.4.14 [noether_lib], page 738): Noether normalization of an ideal(not nessecary homogeneous)
- new library: atkins.lib (Section D.11.2 [atkins_lib], page 1136): the elliptic curve primality test of Atkin
- new library: aksaka.lib (Section D.11.1 [aksaka_lib], page 1134): primality testing after Agrawal, Saxena, Kayal
- new library: arcpoint.lib (Section D.5.2 [arcpoint_lib], page 893): truncations of arcs at a singular point
- new library: resgraph.lib (Section D.8.3 [resgraph_lib], page 1069): visualization of resolution data.
- new library: realrad.lib (Section D.4.20 [realrad_lib], page 790): computation of the real radical over the rational numbers and extensions thereof
- new library: hyperel.lib (Section D.11.4 [hyperel_lib], page 1165): divisors in the jacobian of hyperelliptic curves
- new library: curvepar.lib (Section D.5.4 [curvepar_lib], page 903): space curves
- new library: sagbi.lib (Section D.4.25 [sagbi_lib], page 861): subalgebras bases analogous to Groebner bases for ideals
- new library: surfex.lib (Section D.8.5 [surfex_lib], page 1072): visualizing and rotating surfaces
- new library: cimonom.lib (Section D.4.4 [cimonom_lib], page 671): determines if the toric ideal of an affine monomial curve is a complete intersection.
- Section D.4.26 [sheafcoh_lib], page 864: new experimental functions, in particular Section D.4.26.5 [sheafCohBGG2], page 869
- library `ncall.lib` merged into Section D.2.1 [all_lib], page 526
- library center.lib (`center_lib`) renamed to `central.lib` (Section 7.7.2 [central_lib], page 332)
- Section 7.7.10 [nctools_lib], page 412: new functions for super-commutative algebras (i.e. Section 7.7.10.9 [superCommutative], page 418, Section 7.7.10.20 [IsSCA], page 427, Section 7.7.10.18 [AltVarStart], page 425, Section 7.7.10.19 [AltVarEnd], page 426)
- resolve.lib: blow ups revised (Section D.4.23 [resolve_lib], page 843)
- new algorithms in primdec.lib (Section D.4.18 [primdec_lib], page 779): radical et al.
- improved version of Section 5.1.127 [slimgb], page 218, incorporated into Section 5.1.44 [groebner], page 155, strategy change in groebner

- finvar.lib: the algorithm of Section D.6.1.26 [secondary_char0], page 995 is now used in general in the non-modular case (Section D.6.1 [finvar_lib], page 979)
- finvar.lib: new algorithm for Section D.6.1.27 [irred_secondary_char0], page 997 (Section D.6.1 [finvar_lib], page 979)
- finvar.lib: new function Section D.6.1.30 [irred_secondary_no_molien], page 1000 (Section D.6.1 [finvar_lib], page 979)
- finvar.lib: new functions for computing minimal generating sets of invariant rings of finite groups in the non-modular case: Section D.6.1.5 [invariant_algebra_reynolds], page 982 for finite matrix groups and Section D.6.1.6 [invariant_algebra_perm], page 983 for permutation groups (Section D.6.1 [finvar_lib], page 979)
- operation for sparse matrices improved: multiplication, prune, conversion to module

News in version 3-0-2

The version 3-0-2 is mainly a bug fix release, but it contains also some new features:

- factory, libfac updated for gcc 4.1.x
- configure/speed improved for 64bit architectures
- new library: dmod.lib (Section 7.7.3 [dmod_lib], page 346)
- new library: perron.lib (Section 7.7.11 [perron_lib], page 429)
- improved center.lib: revised implementation, new functions (sa_reduce etc.)
- revised ncalg.lib (Section 7.7.8 [ncalg_lib], page 390): new algebras, U(sl_n) and U(g_2) changed to conform GAP.
- new algorithms in primdec.lib (Section D.4.18 [primdec_lib], page 779): radical et al.
- improved version of Section 5.1.127 [slimgb], page 218, incorporated into Section 5.1.44 [groebner], page 155
- finvar.lib: new algorithm for Section D.6.1.26 [secondary_char0], page 995, which is a drastic improvement (Section D.6.1 [finvar_lib], page 979).
- finvar.lib: new function Section D.6.1.27 [irred_secondary_char0], page 997 (Section D.6.1 [finvar_lib], page 979).
- finvar.lib: new function Section D.6.1.34 [rel_orbit_variety], page 1003, complementing Section D.6.1.35 [relative_orbit_variety], page 1004 (Section D.6.1 [finvar_lib], page 979).
- improved module generator (`modgen`)
- experimental: new type Section 4.1 [bigint], page 70
- more architectures: Solaris on x86/opteron, ...
- build process improved: builds automatically without patches on 64bit architectures

NEWS in version 3-0-1

The version 3-0-1 is mainly a bug fix release, but it contains also some new features:

- NTL upgraded to version 5.4
- new library: absfact.lib Section D.4.1 [absfact_lib], page 652
- new procedures in primdec.lib: Section D.4.18.15 [absPrimdecGTZ], page 787
- new procedures in standard.lib: Section 5.1.152 [weightKB], page 236
- build process improved: new make target `install_all`

- improved version of Section 5.1.127 [slimgb], page 218, incorporated into Section 5.1.44 [groebner], page 155
- arithmetic in Z/pZ for architecture x84_64 improved (by 25 %)

NEWS in version 3-0-0

The version 3-0-0 is the first in the new release series version 3-0.

SINGULAR version 3 has a greater functionality and an improved architecture. Major new features are:

Section A.1.9 [Dynamic modules], page 444

non-commutative extension Chapter 7 [Non-commutative subsystem], page 262

name spaces (Section 4.13 [package], page 111)

easy manipulation of rings (Section 5.1.121 [ringlist], page 211)

improved speed of maps (Section 5.1.136 [subst], page 226, Section 4.9 [map], page 98)

a new algorithm for Groebner base computations: Section 5.1.127 [slimgb], page 218.

improved factorization (integration of NTL, factorization over algebraic extensions, zeroset.lib)

improved gcd of polynomials (including bug fixes)

ports to more architectures including automatic building from source (fink on Mac, ebuild on Gentoo)

more help browsers supported, user extendable (help.cnf)

better support of graded modules

many bugs fixed

Besides theses internal changes, SINGULAR version 3 offers many new features and functionalities (which were partly already incorporated in the 2-1 pre-release series).

Factorizing algorithms revisited (3-0-0)

Starting with version 2-0-4, we use NTL (of Victor Shoup) for factoring univariate polynomials. The multivariate factorization code in libfac/factory now also works over algebraic field extension.

New SINGULAR libraries (3-0-0)

Section D.10.2 [control_lib], page 1119
: algebraic analysis tools for System and Control Theory

Section D.4.13 [mregular_lib], page 733
: procedures for computing the Castelnuovo-Mumford regularity

Section D.4.12 [mprimdec_lib], page 724
: procedures for primary decomposition of modules

Section D.5.8 [gmspoly_lib], page 933, Section D.5.7 [gmssing_lib], page 919
: procedures for the Gauss-Manin connection of a singularity

Section D.4.23 [resolve_lib], page 843, Section D.4.24 [reszeta_lib], page 852
: resolution of singularities and applications

Section D.11.7 [rootsmr_lib], page 1181
: counting the number of real roots

Section D.4.26 [sheafcoh_lib], page 864
: computing cohomology of sheaves and Tate resolution

Section D.7.6 [signcond_lib], page 1055
: computing realizable sign conditions

Section D.11.8 [rootsur_lib], page 1189
: counting number of real roots of univariate polynomial

Section D.4.6 [grwalk_lib], page 681
: Groebner and Pertubation walk

Changes in SINGULAR libraries (3-0-0)

Many procedures were moved into different libraries, the documentation of libraries was generally improved.

All procedures which used to change the current ring return now the newly created ring.

New SINGULAR functions (3-0-0)

Section 5.1.121 [ringlist], page 211
: manipulation of rings

Section 5.1.127 [slimgb], page 218
: a new algorithm to compute Groebner bases

Section 5.2.6 [exportto], page 240
: transfer an identifier to the specified package

Section 5.2.9 [importfrom], page 242
: generate a copy of an identifier from the specified package in the current package

Section 5.1.40 [frwalk], page 153
: fractal walk to change monomial orderings

SINGULAR functions whose syntax/semantics has changed (3-0-0)

Section 5.1.16 [degree], page 139
: outputs degree and dimension instead of returning an int

Section 5.1.1 [attrib], page 127
: new attribute: `rowShift`

Non-commutative Extension PLURAL (3-0-0)

Starting with version 3-0-0, SINGULAR features the non-commutative extension PLURAL. It allows to set and to compute within non-commutative algebras with PBW basis. Among available algorithms are Groebner bases for left modules, syzygies and resolutions.

The following libraries come together with PLURAL:

`center.lib`
: computes generators of the center and centralizer subalgebras up to a given degree resp. up to a given number of generators

Section 7.7.6 [involut_lib], page 383
: determines linear antiautomorphisms (involutions) and automorphisms

Section 7.7.7 [gkdim_lib], page 389
: computes a Gel'fand-Kirillov dimension for modules

Section 7.7.8 [ncalg_lib], page 390
: includes ready-to-use funtions for defining many important non-commutative algebras

Section 7.7.9 [ncdecomp_lib], page 409
: computes a decomposition of a module by its central characters

Section 7.7.8 [ncalg_lib], page 390, Section 7.7.10 [nctools_lib], page 412
: contain numerous useful tools for non-commutative algebras

Section 7.7.12 [qmatrix_lib], page 430
: procedures, related to quantum matrices and minors

Internal Changes (3-0-0)

new mapping code

factory revised
: (gcd, factorizing polynomial over algebraic extension fields,...)

Porting

- SINGULAR is available for ix86-Linux, SunOS-5, IRIX-6, ix86-Win (runs on Windows 95/98/NT4/2000/XP/Vista), FreeBSD, MacOS X, x86_64-Linux (AMD64/Opteron/EM64T), IA64-Linux

8.2 Download instructions

SINGULAR is available as binary program for most common hard- and software platforms. Instructions to download and install SINGULAR can be found at

`http://www.singular.uni-kl.de/download.html`.

Release versions of SINGULAR are also available from our FTP site

`ftp://www.mathematik.uni-kl.de/pub/Math/Singular/`.

To download SINGULAR for a Unix platform

Make sure that you have approximately 20 MByte of free disk space and follow these steps.

1. You need to download two (archive) files:

   `a:` Singular-3-1-1-share.tar.gz contains architecture independent data like documentation and libraries

   `b:` Singular-3-1-1-<uname>.tar.gz contains architecture dependent executables, like the SINGULAR program.

   `<uname>` is a description of the processor and operating system for which SINGULAR is compiled. Choose one of the following:

   `ix86-Linux` PC's running under Linux with a current `libc` version.

`ix86-Win`	PC's running Windows 95/98/NT/2K/XP/Vista which have Cygwin version 1.0 (or higher) already installed. Unless you are familiar with Cygwin, we recommend that you download one of the self-extracting archives as described below.
`SunOS-5`	Sun workstations running Solaris version 5
`sparc64-Linux`	Sun workstations running Linux in 64bit mode
`ix86-SunOS`	PC/Opteron workstations running Solaris version 5
`ppcMac-darwin`	ppc-Macintosh computers running OS X (10.4)
`ix86Mac-darwin`	ix86-Macintosh computers running OS X (10.5)
`x86_64-Linux`	Opteron workstations running Linux in 64bit mode
`IRIX-6`	IRIX workstations running IRIX version 6
`AIX-4`	AIX workstations running AIX version 4

Please contact us if you cannot find an appropriate architecture dependent archive.

2. Simply change to the directory in which you wish to install SINGULAR (usually wherever you install 3rd-party software):

```
cd /usr/local/
```

SINGULAR specific subdirectories will be created in such a way that multiple versions and multiple architecture dependent files of SINGULAR can coexist under the same `/usr/local/` tree.

3. Unpack the archives:

```
gzip -dc Singular-3-1-1-<uname>.tar.gz | tar -pxf -
gzip -dc Singular-3-1-1-share.tar.gz | tar -pxf -
```

4. After unpacking, see the created file `Singular/3-1-1/INSTALL` (which is also located at `ftp://www.mathematik.uni-kl.de/pub/Math/Singular/INSTALL_unix.html` for details on how to finish the installation.

To download SINGULAR for Windows 95/98/NT/2K/XP/Vista

1. Download one of the following self-extracting archives:

`Singular-3-1-1-Small.exe`
: Minimal archive to download. Installs SINGULAR and a minimal set of needed tools/DLLs.

`Singular-3-1-1-Full.exe`
: Complete archive to download. Installs SINGULAR and the XEmacs editor for running ESingular.

2. Double-click (or, execute), the self-extracting archives, and **carefully** follow the instructions given there.
3. In case of problems, consult `http://www.singular.uni-kl.de/WINDOWS/index.html`.

To download SINGULAR for the Macintosh

We recommend `fink` to install `Singular` on a Macintosh running Mac OsX: it supports currently the follwing versions of OsX: 10.4/intel, 10.4/powerpc, 10.4/powerpc (transitional), 10.3. If you would like to install a binary package, Unix installation instructions apply. (See `http://fink.sf.net`)

8.3 Unix installation instructions

The possibilities for installing SINGULAR on a Unix or Linux system range from a purely manual installation over a convenient installation of RPM or DEB packages up to a completely automated installation via a package tool like apt or yum.

For a Linux system with RPM (like Redhat/Fedora, Mandrake/Mandriva, SuSE etc.), or with DEB (like Debian, Ubuntu etc.) one should follow the intructions at `http://www.singular.uni-kl.de/UNIX/`.

The manual installation (for all other Unix systems) is described below:

To install SINGULAR on a Unix platform, you need the following two archives:

`a:` Singular-3-1-1-share.tar.gz contains architecture independent data like documentation and libraries.

`b:` Singular-3-1-1-uname.tar.gz contains architecture dependent executables, like the SINGULAR program.

`uname` is a description of the processor and operating system for which SINGULAR is compiled (e.g, `ix86-Linux`). Please contact us if you cannot find an appropriate architecture dependent archive.

You can obtain these (and other) archives from

ftp://www.mathematik.uni-kl.de/pub/Math/Singular/UNIX/.

To install SINGULAR

Make sure that you have approximately 20 MByte of free disk space and follow these steps.

1. Simply change to the directory in which you wish to install SINGULAR (usually wherever you install 3rd-party software), for example:

   ```
   cd /usr/local
   or
   mkdir install;cd install
   (you do not need root privileges in this case)
   ```

   SINGULAR specific subdirectories will be created in such a way that multiple versions and multiple architecture dependent files of SINGULAR can peaceably coexist under the same `/usr/local` tree.

2. Unpack the archives:

   ```
   gzip -dc <path_to>/Singular-3-1-1-uname.tar.gz | tar -pxf -
   gzip -dc <path_to>/Singular-3-1-1-share.tar.gz | tar -pxf -
   ```

   This creates the directory Singular/3-1-1 with

(sub)directories	which contain
`uname`	`Singular` and `ESingular` executables
`LIB`	SINGULAR libraries (`*.lib` files)
`emacs`	files for the SINGULAR Emacs user interface
`info`	info files of SINGULAR manual
`html`	html files of SINGULAR manual
`doc`	miscellaneous documentation files
`examples`	SINGULAR examples (`*.sing` files)

   For the executable to work, the directory layout must look pretty much like this; the executable will look for "sibling" directories at run-time to figure out where its SINGULAR libraries and on-line documentation files are. These constraints on the local directory layout are necessary

to avoid having to hard-code pathnames into the executables, or require that environment variables be set before running the executable. In particular, you **must not move or copy** the SINGULAR executables to another place, but use soft-links instead.

The following steps are optional:

- Arrange that typing `Singular` at the shell prompt starts up the installed SINGULAR executable.

  If you have root permission, do:

  ```
  ln -s `pwd`/Singular/3-1-1/uname/Singular  /usr/local/bin/Singular-3-1-1
  ln -s `pwd`/Singular/3-1-1/uname/ESingular  /usr/local/bin/ESingular-3-1-1
  ln -s /usr/local/bin/Singular-3-1-1 /usr/local/bin/Singular
  ln -s /usr/local/bin/ESingular-3-1-1 /usr/local/bin/ESingular
  ```

  Otherwise, append the directory `` `pwd`/Singular/3-1-1/uname/ `` to your `$PATH` environment variable. For the `csh` (or, `tcsh`) shell do:

  ```
  set path=(`pwd`/Singular/3-1-1/uname $path)
  ```

  For the `bash` (or, `ksh`) shell do:

  ```
  export PATH=`pwd`/Singular/3-1-1/uname/:$PATH
  ```

  You also might want to adjust your personal start-up files (`~/.cshrc` for `csh`, `~/.tcshrc` for `tcsh`, or `~/.profile` for `bash`) accordingly, so that the `$PATH` variable is set automatically each time you login.

  IMPORTANT: Do *never* move or copy the file `Singular/3-1-1/uname/Singular` to another place, but use soft-links instead.

- If you wish to use any of the following features of SINGULAR, make sure that the respective programs are installed on your system:

Feature	**Requires**
running `ESingular`, or `Singular` within Emacs	Emacs version 20 or higher, or, XEmacs version 20.3 or higher (ESingular is only included in the Linux distribution. On other Unix platforms you can download the SINGULAR emacs lisp files but we give no warranties for specific platforms).
on-line `html` help	any web browser
on-line `info` help	info, or tkinfo texinfo browser programs
TAB completion and history mechanism of ASCII-terminal interface	shared readline library, i.e. `/usr/lib/libreadline.so`
visualization of curves and surfaces	surfer Setup executable for the visualization tool surfer.

  You may download most of these programs from

  `ftp://www.mathematik.uni-kl.de/pub/Math/Singular/utils/`.

- Customize the on-line help system:

  By default, on-line help is displayed in `html` format using the `netscape` program.

  However, this behavior can be customized in several ways using the SINGULAR commands `system("--browser",<browser>)` and `system("--allow-net", 1)` (or, by starting up SINGULAR with the respective command line options).

  In particular, creating the file `Singular/3-1-1/LIB/.singularrc` and putting the SINGULAR command

  ```
  system("--allow-net", 1);
  ```

  in it, allows the on-line help system to fetch its `html` pages from Singular's WWW home site in case its local html pages are not found. That is, you may delete your local html pages, after setting this option.

See also Section 3.1.3 [The online help system], page 15, Section 3.1.6 [Command line options], page 19, and Section 3.1.7 [Startup sequence], page 21, for more details on customizing the on-line help system.

- Add the line

```
* Singular:(singular.hlp).     A system for polynomial computations
```

to your system-wide `dir` file (usually `/usr/info/dir` or `/usr/local/info/dir` and copy or soft-link the file `Singular/3-1-1/info/singular.hlp` to the directory of your `dir` file.
This assures that the SINGULAR manual can be accessed from stand-alone texinfo browser programs such as `info` or `Emacs`. (This is not a prerequisite for using the help system from within `Singular`.)

Troubleshooting

- General: SINGULAR cannot find its libraries or on-line help
  1. Make sure that you have read and/or execute permission the files and directories of the SINGULAR distribution. If in doubt, `cd` to the directory where you unpacked SINGULAR, and do (as root, if necessary):

```
chmod -R a+rX Singular
```

  2. Start up SINGULAR, and issue the command `system("Singular");`. If this does not return the correct and expanded location of the SINGULAR executable, then you found a bug in SINGULAR, which we ask you to report (see below).
  3. Check whether the directories containing the libraries and on-line help files can be found by SINGULAR: If `$bindir` denotes the directory where the SINGULAR executable resides, then SINGULAR looks for library files as follows:
  (0) the current directory
  (1) all dirs of the environment variable SINGULARPATH
  (2) `$bindir/LIB`
  (3) `$bindir/../LIB`
  (4) `/usr/local/Singular/3-1-1/LIB`
  (4) `/usr/local/Singular/LIB`
  The on-line `info` files need to be at `$bindir/../info` and the `html` pages at `$bindir/../html`.

  You can inspect the found library and `info`/`html` directories by starting up SINGULAR with the `--version` option, or by issuing the SINGULAR command `system("--version");`.
- Under previous SuSE-Linux releases, ESINGULAR did not display a prompt: This is due to the very restrictive access rights of `/dev/pty*` of the standard SuSE distribution (starting from version 6.3 on). The problem may still be present when working with the latest SuSE-Linux release. As root, do one of the following: Either

```
chmod 666 /dev/pty*
```

or,

```
chmod g+s $(which emacs)
chgrp tty $(which emacs)
chmod g+s $(which xemacs)
chgrp tty $(which xemacs)
```

- For `ix86-Linux` systems:
Due to some incompatibilities of shared libraries, the start-up of SINGULAR might fail with messages like

```
Can not find shared library ...
```

For DEBIAN systems, try to do `ln -s /usr/lib/libncurses.so /usr/lib/libncurses.so.4`.
If this fails (and on other systems) download and install `Singular-3-1-1-ix86-Linux-static.tar.gz`.

- For `AIX` systems:
  The default `info` program of the system is not GNU's texinfo browser which is used to display the on-line documentation in the `info` format. Therefore, the distribution of the AIX executable already contains the `info` browser program. If you remove this program, make sure that the GNU `info` program is executed if you call '`info`' from your shell.
- For any other troubles:
  Please send an email to singular@mathematik.uni-kl.de and include the header which is displayed by starting up SINGULAR with the `-v` option, and a description of your machine (issue the command `uname -a` on your shell) in your report.

8.4 Windows installation instructions

For the impatient (single file download):

If you have `Cygwin` installed, please read the information below.

Execute the self-extracting installation archive and **carefully** follow the instructions given there. Setup will analyze your system, create the corresponding configuration and give you further tips regarding the installation process. (see http://www.singular.uni-kl.de/WINDOWS/pfwinstall.html)

In case of troubles, see the installed file `/etc/INSTALL` or visit **Singular Forum** at http://www.singular.uni-kl.de/forum.

Installation preliminaries

SINGULAR is a CYGWIN package on MS Windows. You can install it with the Cygwin Installer (Net Install): For a minimal installation, the following set of packages is recommended to be chosen when the Cygwin package browser comes up:

- singular-base, singular-share, singular-help, singular-icons;
- rxvt.

These packages can be complemented by the following, in order to install a full version supporting - among other additional features - the use of SINGULAR under (x)emacs and SINGULAR Surf:

- singular-surf,
- emacs-X11, xemacs-sumo,
- xorg-server, xinit.

Complete installation information for Windows is available at

http://www.singular.uni-kl.de/WINDOWS/index.html.

For any other troubles, please send an email to singular@mathematik.uni-kl.de and include the header which is displayed by starting up SINGULAR with the `-v` option, and a description of your machine and operating system.

8.5 Macintosh installation instructions

Installation of the provided binaries

Do not use the Finder for the installation!

After inserting the CD, open a terminal (under /Applications/Utilities) and type the following commands (you need an Admin Account):
sudo tar -xvzf /Volumes/CDROM/MAC/OSX/Singular-3-0-3-ppcMac-darwin.tar.gz
sudo tar -xvzf /Volumes/CDROM/MAC/OSX/Singular-3-0-3-share.tar.gz
sudo ln -sf /usr/local/Singular/3-0-3/ppcMac-darwin/Singular /usr/local/bin/Singular-3-0-3
The first two commands unpack the tar.gz files and copy the Singular files (binaries and documentation, libs, etc) to /usr/local/Singular/3-0-3.

The final command creates a link in /usr/local/bin so that you can just type Singular-3-0-3 to run SINGULAR in any terminal.

(The expert might wish to read the INSTALL_unix.html file in the UNIX directory for alternatives).

Installation via fink

Fink is a well-known package format for Mac OS X. (see http://fink.sf.net)

There are two packages: singular-doc and singular.

They are available for 10.3 and 10.4 (in fink called 10.4-transitional).

If you have fink installed, choose one of the two possibilities:

type in a shell: `fink selfupdate`
`fink install singular singular-doc`

using the Fink Commander:

Start your Fink Commander

After updating the package list, there should appear "singular" and "singular-doc" as packages

Install it from sources (Fink will do that for you automatically.)

9 Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Table of Contents