CASA User Manual
(Version CASA 2.5)[1]

Ralf Hemmecke and Franz Winkler

Research Institute for Symbolic Computation (RISC-Linz),
Johannes Kepler University
A-4040 Linz, Austria
E-mail: {hemmecke, winkler}@risc.uni-linz.ac.at

October 24, 2000

[1]Supported by SFB F013, Project 1304

Contents

Abstract

CASA is a special-purpose system for computational algebra and constructive algebraic geometry. The system has been developed since 1990. **CASA** is the ongoing product of the Computer Algebra Group at the Research Institute for Symbolic Computation (RISC-Linz), the University of Linz, Austria, under the direction of Prof. Winkler. The system is built on the kernel of the widely used computer algebra system **Maple**.

Chapter 1

Introduction

Constructive methods in algebra and algebraic geometry are gaining more and more importance with the availability of computers and computer algebra softwares. Since its first version (see [GKWW91]), **CASA** has been designed to perform computations and utilize reasoning about algebraic and geometrical objects in classical affine and projective spaces over algebraically closed fields of characteristic zero. More precisely, the field has to be a computable field in the sense of the underlying computer algebra system **Maple**, i.e. all the arithmetic operations have to be available in the system. Usually, the field of computation is the rational numbers $\mathbb{Q}$ or a finite algebraic extension thereof.

Recently, new coding theoretic algorithms have been included. They require computations over finite fields.

Several people have contributed to the system in one way or another (see [Wal88], [Wal91], [GKWW91], [MWW93], [Win96], [Wal93], [Wal93], [CAS94], [Sen94], [MWW95], [Mňu95], [Tra95a], [Tra95b], [MW96], [Tra96], [TW97b], [TW97a], [SW97], [HHSW98], [Win98], [Dei99], [Hem99], [Hac00], [HHW00], [HW00]).

The names of the main contributors are Klaus Aigner, Harald Deischinger, Rüdiger Gebauer, Hong Gu, Michaela Hackl, Ralf Hemmecke (current coordinator), Erik Hillgarter, Michael Kalkbrener, Michal Mňuk, J. Rafael Sendra, Peter Stadelmeyer, Quoc-Nam Tran, Bernhard Wall, Geert Van de Weyer, Franz Winkler (director).

In the system, an algebraic set (a central notion in algebraic geometry) can be represented in four different ways:

- Implicit representation: An algebraic set is the set of common zeros of a system of polynomial equations. To give an algebraic set in implicit form means to give finitely many polynomials.

- Projected representation: As a consequence of the primitive element theorem every irreducible d-dimensional algebraic set in n-dimensional space is, after a suitable linear transformation of coordinates, birationally projectable onto an irreducible d-dimensional algebraic set in $(d+1)$-dimensional space, which can be specified by a single polynomial in $d + 1$ variables. This can be generalized to unmixed-dimensional algebraic sets. An algebraic set in projected form is given by a polynomial and a tuple of rational functions (specifying the birational mapping).
- Parametric representation: Some irreducible algebraic sets can be parametrized by rational functions. An algebraic set in parametric form is given by a tuple of rational functions that parametrizes the algebraic set.
- Representation by places: All algebraic curves can be locally parametrized by a set of power series that are convergent around a point of the curve. An algebraic curve is given by places if for each branch passing through a certain point on the algebraic set a tuple of power series that parametrizes the algebraic set around the point is specified.

The system provides a variety of operations on algebraic sets. As the efficiency of these operations is tightly bound to the way algebraic sets are represented, conversion routines are provided to support various views on one object, to deepen the understanding of its principles, and to speed up algorithms working on algebraic sets. **CASA** also works with the polynomial ideals corresponding to these geometric objects.

The basic operations available in CASA include:

- ideal theoretic operations (union, intersection),
- creating algebraic sets in different representations,
- generating curves of fixed multiplicities at given points,
- intersection, union, and difference of algebraic sets,
- computation of tangent cones and tangent spaces,
- computation of the dimension of an algebraic set,
- decomposition into irreducible components,
- transformations of algebraic sets to hypersurfaces,
- computation of the singularities, genus, neighborhood graphs and adjoint curves of an algebraic curve.

Besides these basic operations, the following more advanced operations are available:

- rational parametrization of rational curves over an optimal extension field of coefficients,
- implicitization of parametrically given algebraic sets,
- Puiseux series expansions,
- multivariate resultants,
- Gröbner bases of ideals and modules,
- Gröbner walk,
- hybrid methods for finding solutions of an arbitrary system of equations,
- computation of rational points on conics,
- offset curves,
- plotting both explicitly and implicitly given curves and surfaces,
- computation of syzygy-bases,
- computation of multivariate polynomial codes and algebraic geometric codes.

The major goal of **CASA** is to provide a comfortable, easy to use, efficient, flexible, and mathematically exact working environment for computational algebra and constructive algebraic geometry where all basic theoretical concepts map easily to available data structures.

CASA is built on the kernel of **Maple** and is fully independent of the operating system; hence, it can be used on every hardware where **Maple** is running.

Chapter 2

Requirements, Distribution And Contact

CASA is independent of the operating system. It is fully written in the **Maple** language.

CASA 2.5 requires **Maple** V release 5. A version for **Maple** 6 is in preparation.

The **CASA** system is freely distributed under the conditions stated in the LICENSE file. In particular we require that each **CASA** user has registered with us and has received a CASA USER KEY from us via email BEFORE **CASA** can be used.

The system can be obtained by anonymous ftp through our download page at http://www.risc.uni-linz.ac.at/software/casa/download.

Please direct any questions, suggestions, and bug report to casa@risc.uni-linz.ac.at.

Chapter 3

CASA: A Quick Tour

3.1 Getting Started

CASA is distributed as a single **Maple** archive of compiled routines and is loaded via the with function after having the language Maple loaded.

```
> with(casa):
```

```
           |__|
           |  |          Welcome to CASA 2.5 for Maple V.5
     |   /\|  |/\
    /=\__|  []  |
   /            \_       Copyright (C) 1990-2000 by Research Institute
   |              \      for Symbolic Computation (RISC-Linz), the
   \   CASA 2.5   |      University of Linz, A-4040 Linz, Austria.
    |             |
   _|     |||     |      For help type '?casa' or '?casa,<topic>'.
__/       |||     |_
```

The code is organized into a number of separate modules. The modules and their major parts are loaded into memory using the readlib facility when they get used for the first time. This significantly cuts the amount of memory occupied by the code.

3.2 Basic Operations

We can define an algebraic set in implicit, parametric or projected representation.

```
> A  := mkImplAlgSet([x*(x^2+y^2-1),x*z,x^2+y^2+z^2-1], [x,y,z]);
```

$$A := \text{Implicit\_Algebraic\_Set}(\\ [x\,(x^2+y^2-1),\, x\,z,\, x^2+y^2+z^2-1],\, [x,\, y,\, z])$$

```
> A2 := mkParaAlgSet([t^2-1,1/(t-1),t+1],[t]);
```

$$A2 := \text{Parametric\_Algebraic\_Set}([t^2-1,\, \frac{1}{t-1},\, t+1],\, [t])$$

```
> A3 := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$A3 := \text{Projected\_Algebraic\_Set}(\\ [[u^2-v^3+v^2],\, [u+v,\, u-v,\, \frac{1}{v}]],\, [u,\, v])$$

We may perform conversion between representations, for example:

1. Birational mapping: We may compute a projection of A. The result is a set of algebraic sets, each given as a hypersurface and a birational mapping to 3D-space. The union of these algebraic sets is equal to A, the intersection of the corresponding ideal is equal to the radical of A. In this case we obtain two one-dimensional circles.

```
> Aproj := toProj(A);
```

$$Aproj := \text{Projected\_Algebraic\_Set}([[x^2+y^2-1],\, [x,\, y,\, 0]],\\ [y,\, x]), \text{Projected\_Algebraic\_Set}(\\ [[z^2-1+x^2],\, [0,\, -x,\, z]],\, [x,\, z])$$

2. Parametrization:

```
> B := mkImplAlgSet([x^2-y^3,z+y+x],[x,y,z]);
```

$$B := \text{Implicit\_Algebraic\_Set}([x^2-y^3,\, z+y+x],\, [x,\, y,\, z])$$

```
> Bpara := toPara(B);
```

$$Bpara := \text{Parametric\_Algebraic\_Set}([\frac{3\,y^2+1+3\,y+y^3}{3\,y-1-3\,y^2+y^3},\\ \frac{y^2+1+2\,y}{y^2-2\,y+1},\, -2\,\frac{y\,(y^2+1+2\,y)}{3\,y-1-3\,y^2+y^3}],\, [y])$$

3. Representation by places (Puiseux expansion)

```
> Bplac := toPlac(B):
> properties(Bplac);
```

"The algebraic set is known to have the following pr \
operties:"

"There are at least 3 terms to show in the Puiseux ex \
pansion"

$$\mathrm{Places\_Algebraic\_Set}([[\frac{1}{8}x^3+\frac{3}{32}x^4+\frac{21}{256}x^5+\frac{5}{64}x^6+\frac{1287}{16384}x^7+\frac{21}{256}x^8+\frac{46189}{524288}x^9+\mathrm{O}(x^{10}),\frac{1}{4}x^2+\frac{1}{8}x^3+\frac{3}{32}x^4+\frac{21}{256}x^5+\frac{5}{64}x^6+\frac{1287}{16384}x^7+\frac{21}{256}x^8+\frac{46189}{524288}x^9+\mathrm{O}(x^{10}),-\frac{1}{4}x^2-\frac{1}{4}x^3-\frac{3}{16}x^4-\frac{21}{128}x^5-\frac{5}{32}x^6-\frac{1287}{8192}x^7-\frac{21}{128}x^8-\frac{46189}{262144}x^9+\mathrm{O}(x^{10})]],[x])$$

3.3 Set And Ideal Operations

The implicit representation of algebraic sets suggests a way of dealing with algebraic sets by working on their corresponding defining ideals. The radical of an ideal generates set-theoretically the same algebraic set (possibly without multiple components).

```
> A := mkImplAlgSet([x*(x^2+y^2-1),x*z,x^2+y^2+z^2-1], [x,y,z]);
```

$$A := \mathrm{Implicit\_Algebraic\_Set}([x\,(x^2+y^2-1),\,x\,z,\,x^2+y^2+z^2-1],\,[x,\,y,\,z])$$

```
> radA := computeRadical(A);
```

$$radA := \mathrm{Implicit\_Algebraic\_Set}([x^2+y^2-1,\,z],\,[x,\,y,\,z]),\ \mathrm{Implicit\_Algebraic\_Set}([x,\,y^2+z^2-1],\,[x,\,y,\,z])$$

During the computation of the radical, the set was split into two components. We can compute the union of these two algebraic sets by computing the intersection (i.e. the least common multiple) of their ideals.

```
> G1 := implUnionLCM(radA);
```

$$G1 := \mathrm{Implicit\_Algebraic\_Set}([x^2+y^2+z^2-1,\,x\,z,\,z\,y^2+z^3-z],\,[x,\,y,\,z])$$

```
> implEqual(A,G1);
```

$$true$$

This gives us the ideal we started with, hence we know that the ideal by which A was given is a radical ideal.

The set-theoretic difference of algebraic sets is computed by determining the quotients of ideals.

```
> H := mkImplAlgSet([x, y^2+z^2-1],[x, y, z]);
```

$$H := \mathrm{Implicit\_Algebraic\_Set}([x,\,y^2+z^2-1],\,[x,\,y,\,z])$$

We can compute the difference of the sets A and H and form the Zariski closure.

```
> implDifference(A,H);
```

$$\text{Implicit\_Algebraic\_Set}([x^2+y^2-1,\ z],\ [x,\ y,\ z])$$

3.4 Computing Dimensions And Irreducible Decomposition

The dimension is an important invariant of algebraic sets. The system **CASA** allows to compute the dimension of algebraic sets given in implicit representation. Let us define an algebraic set in three dimensions.

```
> N := mkImplAlgSet([-y^2*z^2 + 10*y^2*z - y*z^2 - 25*y^2 + 6*y*z
+ x -
> 5*y +1, y^3*z^2 - 10*y^3*z + 25*y^3 - y^2*z + 5*y^2 - y*z + y],
[x, y,
> z]);
```

$$\begin{aligned}&N := \text{Implicit\_Algebraic\_Set}([\\&-y^2\,z^2+10\,y^2\,z-y\,z^2-25\,y^2+6\,y\,z+x-5\,y+1,\\&y^3\,z^2-10\,y^3\,z+25\,y^3-y^2\,z+5\,y^2-y\,z+y],\ [x,\ y,\ z])\end{aligned}$$

N turns out to be a curve.

```
> dimension(N);
```

$$1$$

There are two ways to compute the irreducible decomposition of N. The first one is based on characteristic sets and works directly on the implicit representation of N.

```
> U := decompose(N);
```

$$\begin{aligned}&U := \text{Implicit\_Algebraic\_Set}([1+x,\ y],\ [x,\ y,\ z]),\\&\text{Implicit\_Algebraic\_Set}([-76\,y^3-11\,y^3\,x+y^3\,x^2\\&-15\,x\,y^2-15\,y^2-3\,y-3\,y\,x-1-x, -8\,x^2\,y^2\\&-44\,x-438\,y+65\,z-1064\,y^2+2\,x\,z+x^2\,z\\&-306\,x\,y^2-16\,y\,x^2-108-197\,y\,x-y\,x^3+2\,y^2\,x^3,\\&20\,y\,z+x\,z+5\,z+2\,x^2\,y^2-22\,x\,y^2-152\,y^2-y\,x^2\\&-19\,y\,x-54\,y-4, 4\,z^2+x\,z-23\,z-2\,x^2\,y^2\\&+22\,x\,y^2+152\,y^2-y\,x^2+41\,y\,x+106\,y+16\,x\\&+36],\ [x,\ y,\ z])\end{aligned}$$

The second decomposition algorithm works on algebraic sets in projected form by factoring the defining polynomial of the hypersurface.

```
> decompose(toProj(N));
```

$$\begin{aligned}&\mathrm{Projected\_Algebraic\_Set}([[x^2\,z^2-10\,x^2\,z+25\,x^2+34\,z^2\,x\\&-114\,x\,z-3\,x\,z^3+95\,x-z^5+91+126\,z^2+13\,z^4\\&-59\,z^3-174\,z],[-(95\,x-9\,x\,z-232\,z^2\,x\\&+266\,x\,z^3-104\,z^4\,x+17\,x\,z^5-x\,z^6+25\,x^2-55\,z\\&-11\,z^2-10\,x^2\,z+9\,z^3+49+x^2\,z^2-z^4)/(\%1),-(\\&-10\,z^2\,x+74\,x+25\,x^2-55\,z-11\,z^2-65\,x\,z\\&-10\,x^2\,z+9\,z^3+49+9\,x\,z^3+x^2\,z^2-z^4-z^4\,x)/(\%1\\&),\,z]],\,[x,\,z]),\mathrm{Projected\_Algebraic\_Set}([[1+x],[-(\\&95\,x-9\,x\,z-232\,z^2\,x+266\,x\,z^3-104\,z^4\,x+17\,x\,z^5\\&-x\,z^6+25\,x^2-55\,z-11\,z^2-10\,x^2\,z+9\,z^3+49\\&+x^2\,z^2-z^4)/(\%1),-(-10\,z^2\,x+74\,x+25\,x^2-55\,z\\&-11\,z^2-65\,x\,z-10\,x^2\,z+9\,z^3+49+9\,x\,z^3+x^2\,z^2\\&-z^4-z^4\,x)/(\%1),\,z]],\,[x,\,z])\\&\%1:=-21-56\,z+222\,z^2-257\,z^3+103\,z^4-17\,z^5+z^6\end{aligned}$$

3.5 Algebraic Curves

We define an irreducible projective algebraic curve.

```
> C1 := mkImplAlgSet([-15*y^2*z^3 - 76*y^3*z^2 - z^5 - 3*y*z^4
-
> 15*x*y^2*z^2 - 11*x*y^3*z - x*z^4 - 3*x*y*z^3 + y^3*x^2], [x,y,z],
> ["basespace"="projective"]);
```

$$\begin{aligned}&\mathit{C1}:=\mathrm{Implicit\_Algebraic\_Set}([-15\,y^2\,z^3-76\,y^3\,z^2-z^5\\&-3\,y\,z^4-15\,x\,y^2\,z^2-11\,x\,y^3\,z-z^4\,x-3\,x\,y\,z^3\\&+y^3\,x^2],\,[x,\,y,\,z])\end{aligned}$$

In the course of studying basic properties of $C1$ we will look for singular points. The result of the following function call is a list of all singularities of $C1$ (counted properly). The singular points are collected according to their multiplicities, i.e. first the double points, then the triple points, and so on. Usually, algebraic numbers are introduced in this process.

```
> singularities(C1);
```

$$\begin{aligned}&\mathrm{table}([\\&2=[[\frac{40}{21}\mathrm{RootOf}(5\,\_Z^2+4\,\_Z+89)-\frac{5}{21},\\&-\frac{5}{21}-\frac{2}{21}\mathrm{RootOf}(5\,\_Z^2+4\,\_Z+89),\,1],\,[0,\,1,\,0]]\\&3=[[1,\,0,\,0]]\\&])\end{aligned}$$

Let us take a point on $C1$, $(2:\alpha:1)$, say; α being a root of the irreducible polynomial $94y^3+45y^2+9y+3$.

```
>  alias(alpha=RootOf(94*y^3 + 45*y^2 + 9*y + 3));
```

$$I,\ \alpha$$

The tangent space (i.e. the set of points on all lines which have the intersection multiplicity greater than 1) at this point is the following algebraic set (a line):

```
>  tangSpace(C1,[2,alpha,1]);
```

$$\text{Implicit\_Algebraic\_Set}([-657\,y+48\,z+643\,\alpha\,z+322\,\alpha^2\,z \\ +14\,x\,\alpha-322\,x\,\alpha^2-48\,x],\ [x,\ y,\ z])$$

Remove the alias again.

```
>  alias(alpha=alpha):
```

Since the projective point $(0:1:0)$ is a singular point on $C1$, the tangent space is the whole projective space.

```
>  Ts := tangSpace(C1,[0,1,0]);
```

$$Ts := \text{Implicit\_Algebraic\_Set}([0],\ [x,\ y,\ z])$$

The tangent cone is a linear subspace generating the tangent space. It is given by the following set.

```
>  Tc := tangSpace(C1,[0,1,0],"cone");
```

$$Tc := \text{Implicit\_Algebraic\_Set}([x^2-76\,z^2-11\,x\,z],\ [x,\ y,\ z])$$

The set Tc splits over the algebraic closure of the rational numbers into two lines.

```
>  decompose(Tc,"absolute");
```

$$\text{Implicit\_Algebraic\_Set}(\\ [x+\frac{1}{3}\,z\,\text{RootOf}(\_Z^2-684+33\,\_Z)],\ [x,\ y,\ z]), \\ \text{Implicit\_Algebraic\_Set}(\\ [x+(-11-\frac{1}{3}\,\text{RootOf}(\_Z^2-684+33\,\_Z))\,z], \\ [x,\ y,\ z])$$

It is an interesting question whether coordinates of points on an algebraic set can be expressed as rational functions of some number of parameters. For curves this amounts to representing coordinates of each point in terms of a single parameter. A necessary and sufficient condition is the vanishing of the genus.

```
>  genus(C1);
```

$$0$$

Now we are able to compute a birational mapping of $C1$ to the line. We get a parametric description.

```
> toPara(C1);
```

$$\text{Parametric\_Algebraic\_Set}(\\ [-u^3+3\,u^2-3\,u,\,-\frac{-1+u}{-4+u^2-u},\,1],\,[u])$$

Often it is desirable to determine (systems of) curves passing through specified points with prescribed multiplicity. The system **CASA** offers some functions to perform this task, e.g., passGenCurve.

3.6 Gröbner Bases And Syzygies

Besides the classical Gröbner bases algorithm, **CASA** contains an implementation of Gröbner bases for multivariate polynomial modules and is able to solve systems of linear equations with polynomial coefficients. **CASA** also contains an implementation of the Gröbner Walk algorithm for Gröbner bases conversion.

```
> msolveGB([[x^3 - y, - x + y], [x*y - 1, x^3 - 1], [- x + y^2,
y -
> 1]], [x*y^2 - y^2, - x^5 + x^2 - x + y^2], [x, y], term, tdeg);
```

$$[[y,\,-x^2,\,x],[[x\,y-1+x+y-y^2-x\,y^2+x^3\,y^2-x^4,\\ -x^2+x\,y+y-y^2-y^3-x^3+x\,y^2+x^3\,y,\\ x-2\,y+x^3-x^2\,y+x\,y^2+x^3\,y-x^6]]]$$

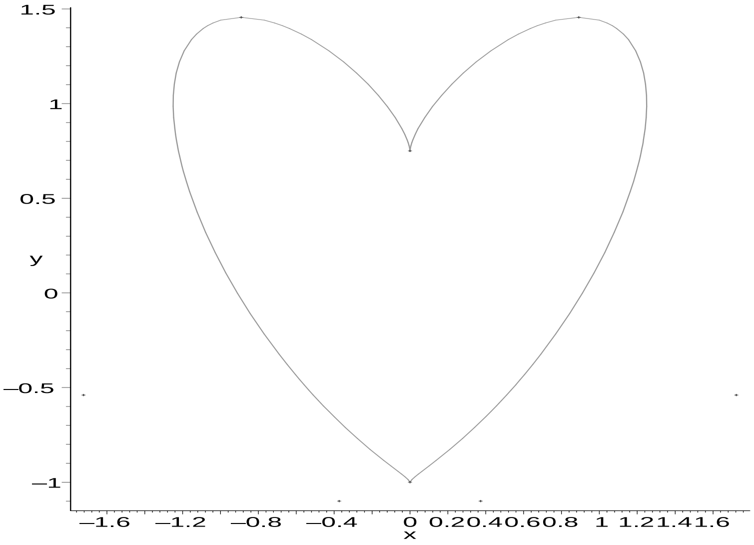
```
> GWalk([t^2 + y^2 + z^2 - 1, t^2 + z^2 - y, t - z], [t, y, z]);
```

$$[2\,z^2-y,\,t-z,\,-4\,z^4+1-2\,z^2]$$

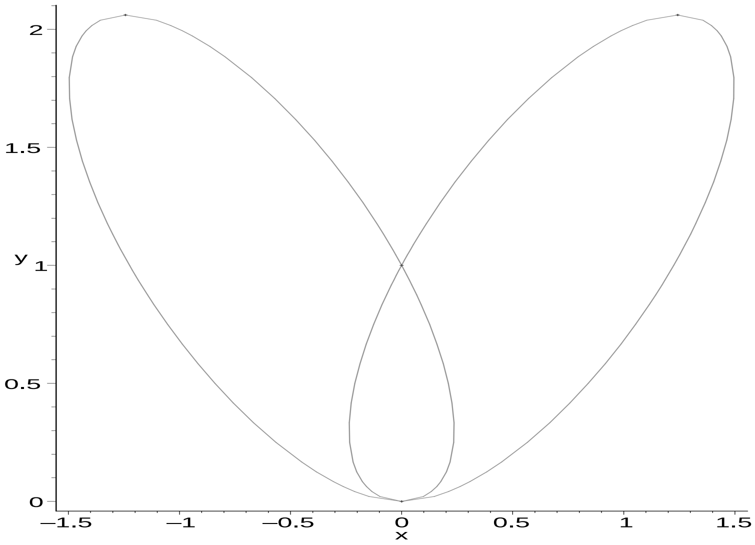
3.7 Scientific Visualization

A powerful feature of **CASA** is its capability to visualize algebraic sets. For the majority of tasks, only singular or other distinguished points of algebraic sets yield some interesting information. These objects have to be treated with care, requiring special analysis. **CASA** makes a thorough effort to obtain correct information about the local topology in the neighborhood of singularities and other critical points while still keeping the algorithm efficient by using hybrid symbolic-numerical methods. Although **Maple** provides a function which is claimed to plot sets in implicit representation, its output for more complicated varieties is by far not satisfactory.

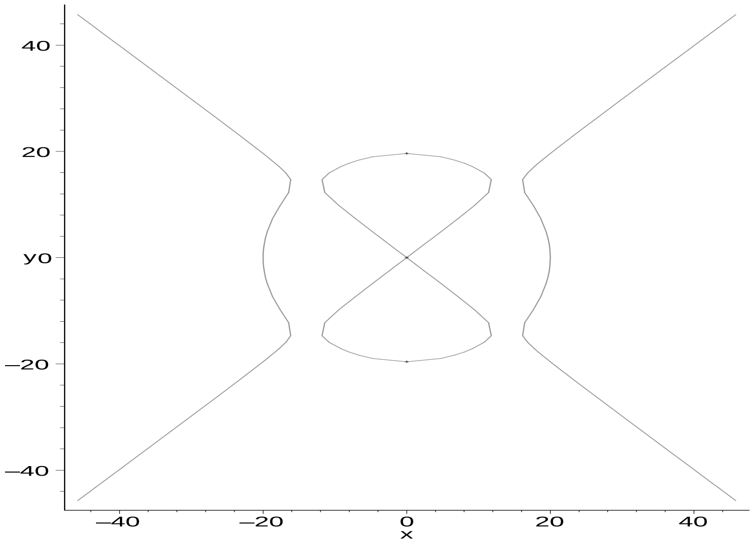
```
> pacPlot(93392896/15625*x^6 + (94359552/625*y^2 + 91521024/625*y
-
> 249088/125)*x^4 + (1032192/25*y^4 - 36864*y^3 - 7732224/25*y^2
-
> 207360*y + 770048/25)*x^2 + 65536*y^6 + 49152*y^5 - 135168*y^4
-
> 72704*y^3 + 101376*y^2 + 27648*y - 27648);
```



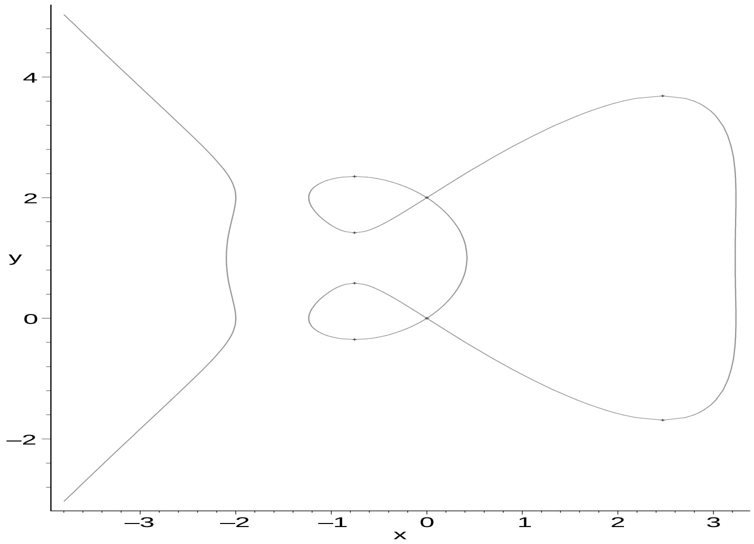
```
> pacPlot(2*x^4 - 3*x^2*y +y^2 - 2*y^3 + y^4);
```



> pacPlot(y^4 - 384*y^2 + 400*x^2 - x^4);



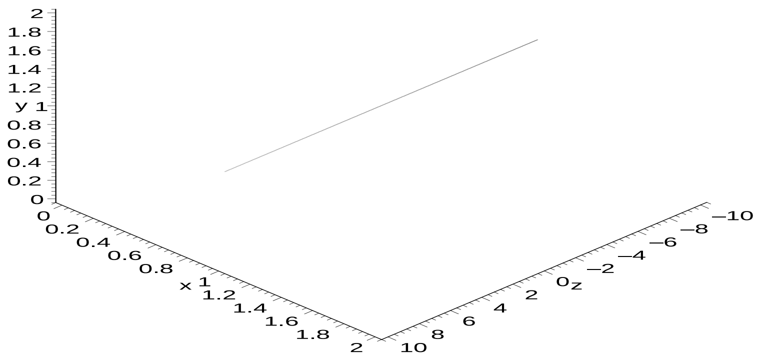
> pacPlot(8*(y^2 - x^2) - 8*(y^3 + x^3) + 2*y^4 + x^5);



The following pictures show the graphs produced by **CASA** for some surface-to-surface intersections taken from literature.

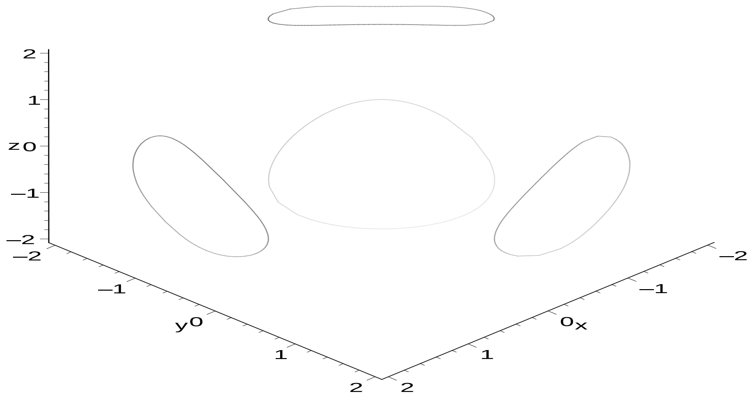
> ssiPlot(x - 1, y - 1, [x, y, z]);

Surface-to-Surface Intersection



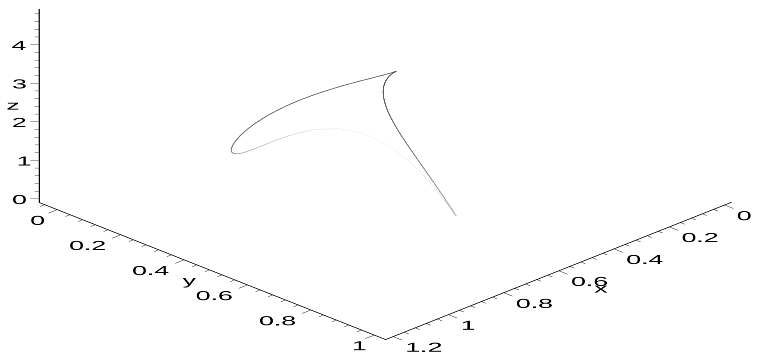

```
>  ssiPlot(x^2 + y^2 + z^2 - 5, x*y*z - 1, [x, y, z]);
```

Surface-to-Surface Intersection



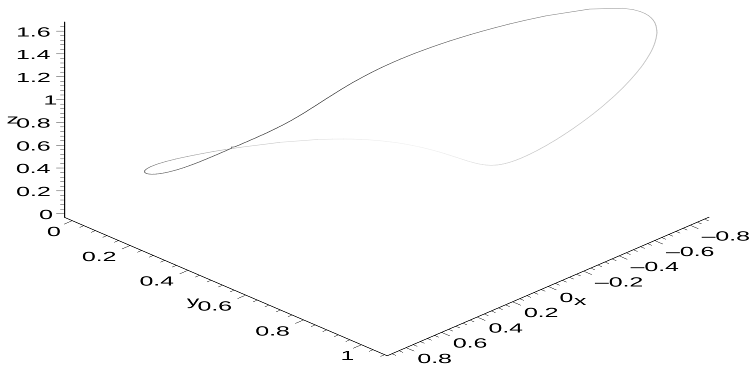

```
>  ssiPlot(z - 2*x^4 - y^4, z - 3*x^2*y + y^2 - x^3, [x, y, z]);
```

Surface-to-Surface Intersection



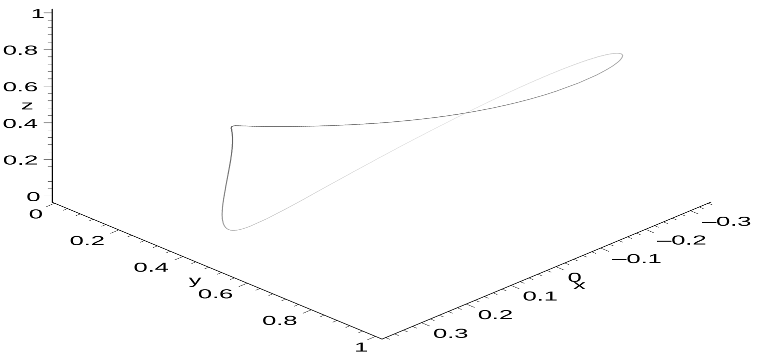

```
>  ssiPlot(z - 2*x^4 - y^4, z^2 - 3*x^2*y - y^3, [x, y, z]);
```

Surface-to-Surface Intersection



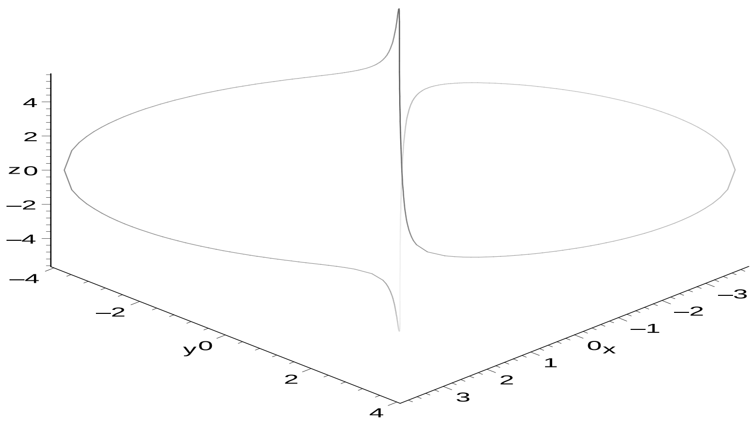

```
>  ssiPlot(z^2 + 3*x^4 - y^4, z + x^3 - 3*x^2*y + y^2 - 2*y^3, [x,
y,
>  z]);
```

Surface-to-Surface Intersection




```
>  ssiPlot(y^4 - 384*y^2 + 400*x^2 - x^4+z^3, x^2 + y^2 + z^2 -30,
[x,
>  y, z]);
```

Surface-to-Surface Intersection



Chapter 4

What Is New

4.1 Help Pages

Help for **CASA** has been revised. In particular we added some type information to the input and output of **CASA** functions.

The help pages are available in 3 different formats: (a) **Maple** online help, (b) printed manual, (c) HTML format on our WEB site http://www.risc.uni-linz.ac.at/software/casa. The contents in each format are the same since they are generated from one source.

Following type faces will be used to expose variables ($exampleVariable$), unevaluated names (*exampleName*), and **CASA** function names (exampleFunction).

4.2 Data Structures

Since the start of **CASA** ten years ago, a data structure has been used to hold the data of an algebraic set in one of 4 different representations (implicit, parametric, or projected representation or a representation by places). The data structure has undergone a complete revision. It now uses its own set of variables to enable **CASA** to store a property list with the algebraic set that will not become unvalid if the user by accident assigns some user variables. This is accomplished by renaming the given variables and parameters of the algebraic set. Each variable is prepended by the string *_casaVariable_* and thus it should be different from any variable on the user level.

We removed any reference to global variables that could interfere with other packages. Apart from the identifiers that are listed in the table *casa* which contains the short names of **CASA** functions, all identifiers in **CASA** start either with *casa* or an additional underscore in front. WE, THEREFORE, FORBID THE ASSIGNMENT TO SUCH NAMES, OTHERWISE WE CAN NOT GUARANTEE THAT **CASA** WORKS PROPERLY.

In several functions additional variables must be given for the result. For example, to give the implicit equations of a parametrized curve, two new variables are needed. One can either give them explicitly or let **CASA** choose new variables.

```
> A := mkParaAlgSet([a, 0], [a]);
```

$$A := \mathrm{Parametric\_Algebraic\_Set}([a, 0], [a])$$

```
> B := toImpl(A, [s,t]);
```

$$B := \mathrm{Implicit\_Algebraic\_Set}([t], [s, t])$$

```
> C := toImpl(A);
```

$$C := \mathrm{Implicit\_Algebraic\_Set}([y], [x, y])$$

CASA chooses new variables according to the following rule. Look for the first variable in the list $casaDefaultVariables$ that has no assigned value and does not clash with the input variables. If the list is exhausted, build a variable name that starts with the letter x followed by a natural number and increase the number until an unassigned variable is found.

The variable $casaDefaultVariables$ is preassigned, but can be reassigned by the user to any list of names.

In working with **Maple** V R5 it became possible to work with strings instead of identifiers. We cleaned up the code in several places, especially for indices in tables **CASA** now uses strings.

4.3 New Functions

There are several new function working with algebraic geometric codes. They will be described in separate chapters below.

As one will notice immediately, **CASA** now shows algebraic sets in different representation more clearly to the eye. This has been achieved by adding the function 'print/_casaAlgebraicSet'.

CASA now more clearly distinguishes between affine and projective space. Usually, **CASA** creates algebraic sets in affine space. If one wants to create an algebraic set in projective space, one must add a parameter.

```
> A := mkImplAlgSet([x^2*z - y^3], [x,y,z],
> ["basespace"="projective"]);
```

$$A := \mathrm{Implicit\_Algebraic\_Set}([x^2\, z - y^3], [x, y, z])$$

One now can use the function toProjective to move a given algebraic set to projective space

```
> B1 := mkImplAlgSet([x^2 - y^3], [x,y]);
```

$$B1 := \text{Implicit\_Algebraic\_Set}([x^2 - y^3], [x, y])$$

```
> B := toProjective(B1, z);
```

$$B := \text{Implicit\_Algebraic\_Set}([x^2\, z - y^3], [x, y, z])$$

or use toAffine to consider only an affine part of an algebraic set.

```
> A1 := toAffine(A, z);
```

$$A1 := \text{Implicit\_Algebraic\_Set}([x^2 - y^3], [x, y])$$

Checking whether an algebraic set lives in affine or projective space can be done by the function isProjective.

```
> isProjective(A);
```

$$\mathit{true}$$

```
> isProjective(A1);
```

$$\mathit{false}$$

And whether two algebraic sets live in the same space can be tested by the function equalBaseSpaces.

```
> equalBaseSpaces(A, B);
```

$$\mathit{true}$$

Another new function is mkAlgSet. It is meant to safe some typing, but also allows to create an algebraic set with just the variables renamed or shifted.

```
> B := mkAlgSet(A, [z=a+x, y=-y], [y, a, x]);
```

$$B := \text{Implicit\_Algebraic\_Set}([y^3 + x^2\, a + x^3], [y, a, x])$$

One could also use this function to consider one variable as a parameter, just shrink the variable list.

```
> C := mkAlgSet(B1,[],[x]);
```

$$C := \text{Implicit\_Algebraic\_Set}([x^2 - y^3], [x])$$

Some access functions to the internal data of an algebraic set have been renamed. Now use generators instead of represent and variableList instead of varlist. A new function parameterList has been added.

```
> generators(C);
```

$$[x^2 - y^3]$$

```
> variableList(C);
```

$$[x]$$

```
> parameterList(C);
```

$$[y]$$

To avoid repeated computations, **CASA** tries to store in the algebraic set structure some additional information, for example, the dimension if it has once been computed. The new function properties shows all stored information about an algebraic set.

```
> properties(C);
```

"No additional information is stored."

Implicit_Algebraic_Set($[x^2 - y^3]$, $[x]$)

```
> dimension(C);
```

0

```
> properties(C);
```

"The algebraic set is known to have the following pr \
operties:"

"A tdeg Groebner basis is: "

$[x^2 - y^3]$

"[] is a list of independent variables"

"It has dimension 0"

Implicit_Algebraic_Set($[x^2 - y^3]$, $[x]$)

To ease conversion from one representation to another the following functions are provided: toImpl, toPara, toPlac, and toProj. They are merely wrapper functions around the various conversion routines that exist in **CASA** for several years. Now it is only necessary to specify the target representation; the source representation can be read from the algebraic set that appears as a parameter.

Algebraic sets that are represented by places are now shown automatically with an approximation up to order 10. One can change this order by the function setPuiseuxExpansion.

There is now a function subresultantChain that computes a subresultant chain and a function rationalPoint that computes a rational point on a conic.

Several auxiliary functions have been added to **CASA**: equalProjectivePoints, pointInAlgSet, homogeneousForm, leadingForm, numberOfTerms, and homogenize.

A few more types are now available in **CASA**: 'type/homogeneousPolynomial', 'type/neighborhoodTree', 'type/casaVariable', and 'type/casaAttributes'.

4.4 Removed Functions

As in the previous version various data is stored into the algebraic set structure once it is computed, however, **CASA** now makes this invisible to the user. Whenever the user asks for some information about an algebraic set, it is first checked whether this information has already been computed and stored. Thus, the functions Attributes, addattr, hasattr, and hasattrib have been removed.

The function repattr has been removed, because algebraic sets will now show directly in which representation they are when they are printed.

The functionality of convertSpace is now achieved through the functions toAffine and toProjective.

The functionality of all conversion functions like convertRep, impl2para, impl2plac, impl2proj, para2impl, para2plac, para2proj, proj2impl, proj2para, and proj2plac is transferred to the functions toImpl, toPara, toPlac, and toProj.

The functions represent, varlist, shAlgSet, and ratpoint have been renamed generators, variableList, properties, and rationalPoint, respectively.

The functionality of reparametrize has been generalized in mkAlgSet.

The type checking function ‘type/projalgpoly‘ has been removed. Use the function ‘type/homogeneousPolynomial‘ instead.

The following functions have been removed without replacement: extNewton, dictionary, geoprove, and dixon.

4.5 Compatibility

CASA in its current version has been upgraded to **Maple** V release 5. Our previous version 2.3 works under **Maple** V release 4 and is available from our download page at http://www.risc.uni-linz.ac.at/software/casa/download. Due to several changes in the **Maple** language, a **CASA** version for **Maple** 6 is not yet available.

Note that due to a stricter type checking the functions mkImplAlgSet, mkParaAlgSet, mkPlacAlgSet, and mkProjAlgSet do not accept all input forms as in the previous version of **CASA**.

The variable ‘$casa/homvars$‘ has been removed. Homogenizing variables will now be provided in a more flexible way through functions and the list stored in $casaDefaultVariables$.

4.6 Miscellaneous

To prevent **CASA** from clashing with other packages, the use of global variables has been cut. Indexing into arrays is now done using strings instead of identifiers.

Several bugs have been removed.

It is now possible to check the version of **CASA**. We introduced 4 new variables that are available globally. The variable ‘$casa/version$‘ contains the version number in a string form, for example, "2.4.3". The other variables will then be: ‘$casa/version/major$‘=2, ‘$casa/version/minor$‘=4, and ‘$casa/version/patch$‘=3.

CASA now supports **Maple**'s *quiet* option. The **CASA** logo will be suppressed when one uses

```
interface(quiet=true): with(casa):
```

to load **CASA**.

4.7 Algebraic Geometric Codes

Algebraic geometric codes have been treated in [Dei99].

We give here a short tutorial on how to work with the corresponding functions implemented in **CASA**. For more details and background information see the above paper and the references therein.

In order to obtain additional information during the computation one can set the infolevel to values from 0 (no information) to 9 (detailed information).

```
> infolevel[`casa/finite`] := 0:
```

At first we construct a Reed-Solomon code of length 15.

We define the code over the line $y = 0$.

```
> F := finiteField(16):
> F["In"](alpha);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)$$

```
> H := finiteCurve(y, F, [x,y,z]);
```

$$H := y$$

We choose the divisor G to be

$$G = -1(0:0:1) + 6(1:0:0)$$

and take the remaining F-rational points as another divisor D.

```
> G := makeDivisor([ [0,0,1], [1,0,0] ], [-1,6]);
```

$$G := Divisor$$

```
> Code := GoppaPrimary(H, "rest", G):
```

This is a code whose parity check matrix looks like this:

```
> print(Code["H"]);
```

$$\left[\alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha, \alpha^2, \alpha^3, \alpha^4, 1, \alpha^5, \alpha^6, \alpha^7, \alpha^8\right]$$

$$\left[\alpha^3, \alpha^5, \alpha^7, \alpha^9, \alpha^{11}, \alpha^{13}, \alpha^2, \alpha^4, \alpha^6, \alpha^8, 1, \alpha^{10}, \alpha^{12}, \alpha^{14}, \alpha\right]$$

$$\left[\alpha^{12}, 1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}, 1, 1, \alpha^3, \alpha^6, \alpha^9\right]$$

$$\left[\alpha^6, \alpha^{10}, \alpha^{14}, \alpha^3, \alpha^7, \alpha^{11}, \alpha^4, \alpha^8, \alpha^{12}, \alpha, 1, \alpha^5, \alpha^9, \alpha^{13}, \alpha^2\right]$$

$$\left[1, \alpha^5, \alpha^{10}, 1, \alpha^5, \alpha^{10}, \alpha^5, \alpha^{10}, 1, \alpha^5, 1, \alpha^{10}, 1, \alpha^5, \alpha^{10}\right]$$

$$\left[\alpha^9, 1, \alpha^6, \alpha^{12}, \alpha^3, \alpha^9, \alpha^6, \alpha^{12}, \alpha^3, \alpha^9, 1, 1, \alpha^6, \alpha^{12}, \alpha^3\right]$$

In order to decode a code word we must first initialize an error locator. Since the above code is defined on a curve of genus 0, we can apply the Skorobotatov-Vladut algorithm.

Since we do not have a one point code here, we must specify a divisor A. The space $L(A)$ is used to find an error locator whereas $L(G - A)$ is used as the space of test functions.

```
> A := makeDivisor([ [1,0,0] ], [3]):
> SV := GoppaPrepareSV(Code, A):
```

Now let us encode the following message.

```
> a := [1, alpha^14, alpha^10, alpha^8, 0, alpha^9, alpha^4,
> alpha^11, alpha^4];
```

$$a := [1, \alpha^{14}, \alpha^{10}, \alpha^8, 0, \alpha^9, \alpha^4, \alpha^{11}, \alpha^4]$$

```
> c := GoppaEncode(a, Code);
```

$$c := \left[1, \alpha^{14}, \alpha^{10}, \alpha^8, 0, \alpha^9, \alpha^4, \alpha^{11}, \alpha^4, \alpha^6, \alpha^3, \alpha^9, \alpha^{14}, \alpha^2, 0\right]$$

Now let us add some errors to the code word c. Our code has minimal distance 7, so it should be no problem if we add 3 errors (in places 1, 7, and 8).

```
> w := c: w[1] := w[1] + alpha: w[7] := w[7] + alpha:
> w[8] := w[8] + alpha^3:
```

To decode the code word again, we need the *Code* and the error locator SV.

```
> b := GoppaDecode(w, Code, SV);
```

$$b := [1, \alpha^{14}, \alpha^{10}, \alpha^{8}, 0, \alpha^{9}, \alpha^{4}, \alpha^{11}, \alpha^{4}]$$

We obtain back the message we started with.

```
> evalm(b-a);
```

$$[0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Let us consider the Hermitian code defined over the smooth curve $x^5+y^4z+yz^4$ over the Galois field with 16 elements. This is a code of length 64.

```
> H := finiteCurve(x^5+y^4*z+y*z^4, F, [x,y,z]);
```

$$H := x^5 + y^4\,z + y\,z^4$$

```
> `casa/finite/Curve/genus`(H);
```

$$6$$

Now we can already construct the primary Goppa code. The divisor D is taken as the sum of the affine F-rational points. The third argument specifies that the divisor G should be taken as 25 times the single point at infinity.

```
> Code := GoppaPrimary(H, "affine", 25):
```

To decode, we must initialize an error locator. Since this code is defined on a curve of genus greater than zero, we can use the Duursma-algorithm.

The Duursma-algorithm is implemented only for one point codes, so no extra arguments are necessary. It is sufficient to specify which code is used.

```
> Du := GoppaPrepareDu(Code):
```

Let us take the following message.

```
> a := [alpha^8, alpha^5, 1, alpha^10, 0, alpha^9, alpha^4,
> alpha^7, 1, 1, alpha^14, alpha^10, alpha^8, 0, alpha^9,
> alpha^4, alpha^11, alpha^4, alpha^10, alpha^4, alpha^5,
> alpha^13, alpha^9, alpha^12, alpha^12, alpha^4, alpha^13,
> alpha^5, 1, alpha^5, alpha^6, alpha^8, alpha^3, alpha^2,
> alpha^9, 1, alpha^11, alpha^3, alpha^4, alpha^6, alpha^10,
> alpha, alpha^4, 0
> ];
```

$$\begin{aligned}&a := [\alpha^{8}, \alpha^{5}, 1, \alpha^{10}, 0, \alpha^{9}, \alpha^{4}, \alpha^{7}, 1, 1, \alpha^{14}, \alpha^{10}, \alpha^{8}, 0, \alpha^{9}, \alpha^{4},\\ &\alpha^{11}, \alpha^{4}, \alpha^{10}, \alpha^{4}, \alpha^{5}, \alpha^{13}, \alpha^{9}, \alpha^{12}, \alpha^{12}, \alpha^{4}, \alpha^{13}, \alpha^{5}, 1,\\ &\alpha^{5}, \alpha^{6}, \alpha^{8}, \alpha^{3}, \alpha^{2}, \alpha^{9}, 1, \alpha^{11}, \alpha^{3}, \alpha^{4}, \alpha^{6}, \alpha^{10}, \alpha, \alpha^{4}, 0\\ &]\end{aligned}$$

Encoding is performed by a simple matrix multiplication.

```
> c := GoppaEncode(a, Code);
```

$$c := \Big[\alpha^8, \alpha^5, 1, \alpha^{10}, 0, \alpha^9, \alpha^4, \alpha^7, 1, 1, \alpha^{14}, \alpha^{10}, \alpha^8, 0, \alpha^9, \alpha^4,$$
$$\alpha^{11}, \alpha^4, \alpha^{10}, \alpha^4, \alpha^5, \alpha^{13}, \alpha^9, \alpha^{12}, \alpha^{12}, \alpha^4, \alpha^{13}, \alpha^5, 1,$$
$$\alpha^5, \alpha^6, \alpha^8, \alpha^3, \alpha^2, \alpha^9, 1, \alpha^{11}, \alpha^3, \alpha^4, \alpha^6, \alpha^6, \alpha^{10}, \alpha^{10},$$
$$1, \alpha, \alpha^4, \alpha^{12}, \alpha^{12}, 0, \alpha^2, \alpha^6, \alpha^{10}, \alpha^8, \alpha^7, \alpha^{13}, \alpha^{12}, \alpha^{14},$$
$$\alpha^{11}, \alpha^8, \alpha^9, \alpha^{14}, \alpha^9, \alpha^4, \alpha^3\Big]$$

Since the code has minimal distance of at least 15, we can correct up to 7 errors. Let us add some randomness.

```
> w := c:
> w[20] := w[20] + alpha^7:
> w[21] := w[21] + alpha^10:
> w[23] := w[23] + alpha^5:
> w[35] := w[35] + alpha:
> w[52] := w[52] + alpha^7:
> w[56] := w[56] + alpha:
> w[58] := w[58] + alpha^14:
```

In order to decode the received word, the code and the error locator must be given. For the entries of order $s = 26$ we really need the voting process. This is not the typical case, however, it might take some tries to find an example where not all error locator candidates give the same estimate. (Detailed information about the computation can be optained by increasing the infolevel.)

```
> b := GoppaDecode(w, Code, Du);
```

$$b := \Big[\alpha^8, \alpha^5, 1, \alpha^{10}, 0, \alpha^9, \alpha^4, \alpha^7, 1, 1, \alpha^{14}, \alpha^{10}, \alpha^8, 0, \alpha^9, \alpha^4,$$
$$\alpha^{11}, \alpha^4, \alpha^{10}, \alpha^4, \alpha^5, \alpha^{13}, \alpha^9, \alpha^{12}, \alpha^{12}, \alpha^4, \alpha^{13}, \alpha^5, 1,$$
$$\alpha^5, \alpha^6, \alpha^8, \alpha^3, \alpha^2, \alpha^9, 1, \alpha^{11}, \alpha^3, \alpha^4, \alpha^6, \alpha^{10}, \alpha, \alpha^4, 0\Big]$$

We obtain back the message we started with.

```
> evalm(b-a);
```

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

4.8 Multivariate Polynomial Codes

Multivariate polynomial codes have been treated in [Hac00].

We give here a short tutorial on how to work with the corresponding functions implemented in **CASA**. For more details and background information see the above paper and the references therein.

Since there is a lot of computation to be done with polynomials over finite fields, several functions are supplied for the basic operations of polynomials in two variables x and y over a finite field.

Generate the field of coefficients of the polynomials;

```
> F := finiteField(16):
```

Generate a polynomial in RootOf form

```
> g := InPolynomial(alpha*x + alpha^2*x^2*y, F);
```

$$g := \mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)\, x + \mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)^2\, x^2\, y$$

Substitute $(x,\, y)$ by (1, alpha).

```
> SubsPolynomial([1, alpha], g, F);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)^3 + \mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)$$

Find the roots of g over the finite field F. Print these roots in 'nice' alpha form.

```
> map(F["mapOut"], PolynomialRoots(g, F));
```

$$\begin{aligned}&[[0,\, 0],\, [0,\, 1],\, [0,\, \alpha],\, [0,\, \alpha^2],\, [0,\, \alpha^3],\, [0,\, \alpha^4],\, [0,\, \alpha^5],\, [0,\, \alpha^6],\\ &[0,\, \alpha^7],\, [0,\, \alpha^8],\, [0,\, \alpha^9],\, [0,\, \alpha^{10}],\, [0,\, \alpha^{11}],\, [0,\, \alpha^{12}],\\ &[0,\, \alpha^{13}],\, [0,\, \alpha^{14}],\, [1,\, \alpha^{14}],\, [\alpha,\, \alpha^{13}],\, [\alpha^2,\, \alpha^{12}],\\ &[\alpha^3,\, \alpha^{11}],\, [\alpha^4,\, \alpha^{10}],\, [\alpha^5,\, \alpha^9],\, [\alpha^6,\, \alpha^8],\, [\alpha^7,\, \alpha^7],\\ &[\alpha^8,\, \alpha^6],\, [\alpha^9,\, \alpha^5],\, [\alpha^{10},\, \alpha^4],\, [\alpha^{11},\, \alpha^3],\, [\alpha^{12},\, \alpha^2],\\ &[\alpha^{13},\, \alpha],\, [\alpha^{14},\, 1]]\end{aligned}$$

Generate a polynomial h.

```
> h := InPolynomial(alpha^2*x*y + 1, F);
```

$$h := \mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)^2\, x\, y + 1$$

Find the common roots of g and h.

```
> map(F["mapOut"], PolynomialRoots([g, h], F));
```

$$[]$$

Two-dimensional BCH codes can be constructed by the function BCH2. The arguments required are the domain of the code, the parameter δ and the finite field. The function CyclicEncode can be used for encoding. The two arguments required are the message which should be encoded and the code. The function BCHDecode can be used for decoding. Its parameters are the received word and the code.

There is a function GoppaPrimary to construct primary Goppa codes. One-point algebraic geometric codes are special primary Goppa codes, therefore you can use this method to construct a 1-point algebraic geometric code. The arguments required are the curve C, and two divisors D and $G = mQ$.

Encoding can be done by GoppaEncode. Its parameters are the message and the code.

For decoding, one has to initialize the method one would like to use for decoding. GoppaPrepareSa prepares all preprocessing steps to use Sakata's approach to decoding. Its argument is the code and its result is a table of the results of all the computation which has to be done only once for a given code. GoppaPrepareDu initializes Duursma's method and if the genus of the curve is low one can also use GoppaPrepareSV to initialize the basic algorithm, which doesn't use majority voting for unknown syndroms.

After this initialization one can use GoppaDecode or SakataDecode to decode a received word. The parameters of this methods are the code, the result of the initialization, and the received word.

Three example sessions will follow. In each session a code is constructed, a message encoded, some errors are added and finally the received word is decoded.

The first example is a Reed-Solomon code over F_{16} with designed minimum distance 5. We force the generator polynomial to vanish on 4 subsequent powers of α, namely 1, α, α^2, α^3.

This code can be regarded as a 2-dimensional BCH-code of domain (1,15) and δ = (4,0).

```
> infolevel[`casa/finite`] := 11:
> F := finiteField(16):
> C := BCH2(1,15, [4,0], F):
```

"Common roots" = [[0, 0], [0, 1], [0, 2], [0, 3]]

$H =$

$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

$\big[1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11},$
$\alpha^{12}, \alpha^{13}, \alpha^{14}\big]$

$\big[1, \alpha^2, \alpha^4, \alpha^6, \alpha^8, \alpha^{10}, \alpha^{12}, \alpha^{14}, \alpha, \alpha^3, \alpha^5, \alpha^7,$
$\alpha^9, \alpha^{11}, \alpha^{13}\big]$

$\big[1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}, 1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}, 1, \alpha^3, \alpha^6$
$, \alpha^9, \alpha^{12}\big]$

$$G = \begin{bmatrix} 1,0,0,0,0,0,0,0,0,0,0,\alpha^6,\alpha^9,\alpha^2,\alpha^6 \\ 0,1,0,0,0,0,0,0,0,0,0,\alpha^{12},\alpha^8,\alpha^5,\alpha^7 \\ 0,0,1,0,0,0,0,0,0,0,0,\alpha^{13},1,\alpha^5,\alpha^{11} \\ 0,0,0,1,0,0,0,0,0,0,0,\alpha^2,\alpha^{11},\alpha^7,\alpha^6 \\ 0,0,0,0,1,0,0,0,0,0,0,\alpha^{12},\alpha^9,\alpha^{12},\alpha^2 \\ 0,0,0,0,0,1,0,0,0,0,0,\alpha^8,\alpha^8,\alpha^{14},\alpha^{11} \\ 0,0,0,0,0,0,1,0,0,0,0,\alpha^2,\alpha^9,\alpha^3,\alpha^3 \\ 0,0,0,0,0,0,0,1,0,0,0,\alpha^9,\alpha^7,\alpha^8,\alpha^{11} \\ 0,0,0,0,0,0,0,0,1,0,0,\alpha^2,\alpha^8,1,\alpha^{10} \\ 0,0,0,0,0,0,0,0,0,1,0,\alpha,\alpha^{11},\alpha^{11},\alpha^{12} \\ 0,0,0,0,0,0,0,0,0,0,1,\alpha^3,\alpha^{11},1,\alpha^9 \end{bmatrix}$$

$$[n,\, k] = 15,\, 11$$

$$\text{“Extension Field”} = 16$$

One can easily obtain some information about the code.

```
> eval(C["Delta"]);
```

$$[4,\, 0]$$

```
> eval(C["k"]);
```

$$11$$

```
> eval(C["Roots"]);
```

$$[[0,\, 0],\, [0,\, 1],\, [0,\, 2],\, [0,\, 3]]$$

```
> eval(C["ExtField"]["seq"]);
```

$$[0,\, 1,\, \alpha,\, \alpha^2,\, \alpha^3,\, \alpha^4,\, \alpha^5,\, \alpha^6,\, \alpha^7,\, \alpha^8,\, \alpha^9,\, \alpha^{10},\, \alpha^{11},\, \alpha^{12},\, \alpha^{13},\, \alpha^{14}]$$

Now encode the message and make two errors.

```
> a := [seq(1, i=1..11)];
```

$$a := [1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1]$$

```
> c := CyclicEncode(a, C);
```

$$c := [1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, \alpha^{14},\, 0,\, \alpha^7,\, \alpha^{10}]$$

```
> w := evalm(c):
> w[7] := w[7] + alpha^5:
> w[13] := w[13] + alpha^8:
> print(w);
```

$$[1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1+\alpha^5,\, 1,\, 1,\, 1,\, 1,\, \alpha^{14},\, \alpha^8,\, \alpha^7,\, \alpha^{10}]$$

Decode the message.

```
> b := BCHDecode(w, C);
```

$$\text{“Syndrom array”} =$$

$$\left[\alpha^9,\, \alpha^{13},\, 0,\, \alpha,\, ?_{1,5},\, ?_{1,6},\, ?_{1,7},\, ?_{1,8},\, ?_{1,9},\, ?_{1,10},\, ?_{1,11},\, ?_{1,12},\, ?_{1,13},\, ?_{1,14},\, ?_{1,15}\right]$$

$$\text{“Sakata's algorithm:”}$$
$$\text{“u”} = [0,\, 0],\ \text{“F”} = [y,\, x],\ \text{“G”} = [1]$$
$$\text{“u”} = [1,\, 0],\ \text{“F”} = [y,\, x+1],\ \text{“G”} = [1]$$
$$\text{“u”} = [0,\, 1],\ \text{“F”} = [y+\alpha^4,\, x+1],\ \text{“G”} = [1]$$
$$\text{“u”} = [2,\, 0],\ \text{“F”} = [y+\alpha^4,\, x+1],\ \text{“G”} = [1]$$
$$\text{“u”} = [1,\, 1],\ \text{“F”} = [y+\alpha^4,\, x+1],\ \text{“G”} = [1]$$
$$\text{“u”} = [0,\, 2],\ \text{“F”} = [\alpha^4\, y+\alpha^8+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$
$$\text{“u”} = [3,\, 0],\ \text{“F”} = [\alpha^4\, y+\alpha^8+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$
$$\text{“u”} = [2,\, 1],\ \text{“F”} = [\alpha^4\, y+\alpha^8+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$
$$\text{“u”} = [1,\, 2],\ \text{“F”} = [\alpha^4\, y+\alpha^8+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$
$$\text{“u”} = [0,\, 3],\ \text{“F”} = [\alpha^{14}\, y+\alpha^3+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$
$$\text{“u”} = [4,\, 0],\ \text{“F”} = [\alpha^{14}\, y+\alpha^3+y^2,\, x+1],\ \text{“G”} = [y+\alpha^4]$$

“Minimal set generating the syndromarrayup to the \
b” = [4, 0]

$$\text{“F”} = [\alpha^{14}\, y+\alpha^3+y^2,\, x+1]$$
$$\text{“common roots of F”} = [[0,\, 6],\, [0,\, 12]]$$
$$\text{“error vector”} = [0,\, 0,\, 0,\, 0,\, 0,\, 0,\, \alpha^5,\, 0,\, 0,\, 0,\, 0,\, 0,\, \alpha^8,\, 0,\, 0]$$

“sent codeword” =
$[1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, \alpha^{14},\, 0,\, \alpha^7,\, \alpha^{10}]$

$$b := [1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1,\, 1]$$

Obviously, this is the message that we started with.

In the next example, we want to constract an at least 2-error correcting code. We choose $\delta = (4{,}0)$ and domain (7,7) over the finite field F_4

```
> infolevel[‘casa/finite‘] := 10:
> F := finiteField(4):
> C := BCH2(7, 7, [4,0], F):
```

“Common roots” = [[0, 0], [1, 0], [4, 0], [2, 0], [0, 1], [0, 4],
[0, 2], [1, 1], [4, 4], [2, 2], [3, 0], [5, 0], [6, 0], [2, 1],
[1, 4], [4, 2], [1, 2], [4, 1], [2, 4], [0, 3], [0, 5], [0, 6]
]

$$[n,\, k] = 49,\, 27$$
$$\text{“Extension Field”} = 64$$

Now encode the message and make two errors.

```
> a := [seq(1, i=1..27)];
```

$a := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

```
> c := CyclicEncode(a, C);
```

$c := [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

```
> w := evalm(c):
> w[1] := w[1] + alpha^2:
> w[19] := w[19] + alpha:
> print(w);
```

$$\left[1 + \alpha^2, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1 + \alpha, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0\right]$$

Decode the message.

```
> b := BCHDecode(w, C);
```

$$\text{“Syndrom array”} = \begin{bmatrix} 1 & \alpha^{44} & \alpha^{11} & \alpha^4 & \alpha^{50} & \alpha^{16} & \alpha \\ \alpha^{50} & \alpha^{16} & \alpha & ?_{2,4} & \alpha^{44} & ?_{2,6} & ?_{2,7} \\ \alpha^{44} & \alpha^{11} & \alpha^4 & ?_{3,4} & \alpha^{16} & ?_{3,6} & ?_{3,7} \\ \alpha^{16} & ?_{4,2} & ?_{4,3} & ?_{4,4} & ?_{4,5} & ?_{4,6} & ?_{4,7} \\ \alpha^{11} & \alpha^4 & \alpha^{50} & ?_{5,4} & \alpha & ?_{5,6} & ?_{5,7} \\ \alpha & ?_{6,2} & ?_{6,3} & ?_{6,4} & ?_{6,5} & ?_{6,6} & ?_{6,7} \\ \alpha^4 & ?_{7,2} & ?_{7,3} & ?_{7,4} & ?_{7,5} & ?_{7,6} & ?_{7,7} \end{bmatrix}$$

$$\text{“Sakata's algorithm:”}$$

$$\text{“u”} = [0, 0],\ \text{“F”} = [y, x],\ \text{“G”} = [1]$$

$$\text{“u”} = [1, 0],\ \text{“F”} = [y, x + \alpha^{50}],\ \text{“G”} = [1]$$

$$\text{“u”} = [0, 1],\ \text{“F”} = [y + \alpha^{44}, x + \alpha^{50}],\ \text{“G”} = [1]$$

$$\text{“u”} = [2, 0],\ \text{“F”} = [y + \alpha^{44}, \alpha^{50}\, x + \alpha^{54} + x^2],\ \text{“G”} = [x + \alpha^{50}]$$

$$\text{“u”} = [1, 1],\ \text{“F”} = [\alpha^{27}\, x + y + \alpha^{18}, \alpha^{50}\, x + \alpha^{54} + x^2],\ \text{“G”} = [x + \alpha^{50}]$$

$$\text{“u”} = [0, 2],\ \text{“F”} = [\alpha^{27}\, x + y + \alpha^{18}, \alpha^{50}\, x + \alpha^{54} + x^2],\ \text{“G”} = [x + \alpha^{50}]$$

“u” = [3, 0], “F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$,
“G” = $[x + \alpha^{50}]$

“u” = [2, 1], “F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$,
“G” = $[x + \alpha^{50}]$

“u” = [1, 2], “F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$,
“G” = $[x + \alpha^{50}]$

“u” = [0, 3], “F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$,
“G” = $[x + \alpha^{50}]$

“u” = [4, 0], “F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$,
“G” = $[x + \alpha^{50}]$

“Minimal set generating the syndromarrayup to the \
b” = [4, 0]

“F” = $[\alpha^{27}\,x + y + \alpha^{18},\ \alpha^{27}\,x + \alpha^{18} + x^2]$

“common roots of F” = [[0, 0], [2, 4]]

“error vector” = $[\alpha^2$, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, α, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

“sent codeword” = [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

b := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Again, this is the message that we started with.

As the last example, we take an Hermitian code defined over the curve $H_4 = V(x^5 + y^4 z + y z^4)$ over the field F_{16}.

This is a 7-error correcting code of length 64. We will show how to decode it using Sakata's approach.

```
> infolevel[‘casa/finite‘] := 9:
> F := finiteField(16):
> F["In"](alpha);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z^3 + 1)$$

```
> H4 := finiteCurve(x^5 + y^4 * z + y * z^4, F);
```

$$H4 := x^5 + y^4\,z + y\,z^4$$

```
> g := ‘casa/finite/Curve/genus‘(H4);
```

$$g := 6$$

```
> C := GoppaPrimary(H4, "affine", 25);
```

$$G = +25\ (0 : 1 : 0)$$

$$\begin{aligned}
D\ = &[\alpha^{12}, \alpha^{11}, 1]\ + [\alpha^{9}, \alpha^{7}, 1]\ + [\alpha^{6}, \alpha^{13}, 1] \\
&+ [\alpha^{3}, \alpha^{13}, 1]\ + [\alpha^{5}, \alpha^{9}, 1]\ + [\alpha^{6}, \alpha^{11}, 1] \\
&+ [\alpha^{10}, \alpha^{12}, 1]\ + [\alpha^{11}, \alpha^{2}, 1]\ + [0, \alpha^{10}, 1] \\
&+ [\alpha^{10}, \alpha^{3}, 1]\ + [\alpha^{10}, \alpha^{4}, 1]\ + [\alpha^{9}, \alpha^{14}, 1] \\
&+ [\alpha^{4}, \alpha^{12}, 1]\ + [\alpha^{10}, \alpha, 1]\ + [\alpha^{5}, \alpha^{6}, 1] \\
&+ [\alpha^{9}, \alpha^{11}, 1]\ + [\alpha^{4}, \alpha, 1]\ + [\alpha^{3}, \alpha^{14}, 1] \\
&+ [\alpha^{9}, \alpha^{13}, 1]\ + [\alpha^{8}, \alpha^{9}, 1]\ + [\alpha^{8}, \alpha^{8}, 1] \\
&+ [\alpha^{7}, \alpha, 1]\ + [0, 0, 1]\ + [\alpha^{6}, \alpha^{14}, 1] \\
&+ [\alpha^{8}, \alpha^{6}, 1]\ + [\alpha^{4}, \alpha^{3}, 1]\ + [\alpha, \alpha^{12}, 1] \\
&+ [\alpha^{3}, \alpha^{7}, 1]\ + [\alpha, \alpha^{4}, 1]\ + [\alpha^{4}, \alpha^{4}, 1] \\
&+ [\alpha^{2}, \alpha^{9}, 1]\ + [1, \alpha^{7}, 1]\ + [\alpha^{3}, \alpha^{11}, 1] \\
&+ [1, \alpha^{11}, 1]\ + [\alpha^{2}, \alpha^{2}, 1]\ + [1, \alpha^{13}, 1] \\
&+ [\alpha^{2}, \alpha^{8}, 1]\ + [\alpha^{2}, \alpha^{6}, 1]\ + [\alpha, \alpha, 1] \\
&+ [1, \alpha^{14}, 1]\ + [0, 1, 1]\ + [0, \alpha^{5}, 1] \\
&+ [\alpha^{14}, \alpha^{9}, 1]\ + [\alpha^{14}, \alpha^{8}, 1]\ + [\alpha^{7}, \alpha^{4}, 1] \\
&+ [\alpha^{14}, \alpha^{6}, 1]\ + [\alpha^{7}, \alpha^{12}, 1]\ + [\alpha^{13}, \alpha, 1] \\
&+ [\alpha^{7}, \alpha^{3}, 1]\ + [\alpha^{13}, \alpha^{12}, 1]\ + [\alpha^{13}, \alpha^{3}, 1] \\
&+ [\alpha^{14}, \alpha^{2}, 1]\ + [\alpha^{13}, \alpha^{4}, 1]\ + [\alpha^{11}, \alpha^{6}, 1] \\
&+ [\alpha^{5}, \alpha^{2}, 1]\ + [\alpha, \alpha^{3}, 1]\ + [\alpha^{12}, \alpha^{13}, 1] \\
&+ [\alpha^{11}, \alpha^{9}, 1]\ + [\alpha^{11}, \alpha^{8}, 1]\ + [\alpha^{5}, \alpha^{8}, 1] \\
&+ [\alpha^{6}, \alpha^{7}, 1]\ + [\alpha^{12}, \alpha^{14}, 1]\ + [\alpha^{8}, \alpha^{2}, 1] \\
&+ [\alpha^{12}, \alpha^{7}, 1]
\end{aligned}$$

$$L(G) = \mathrm{span}(1, \frac{y}{z}, \frac{y^2}{z^2}, \frac{y^3}{z^3}, \frac{y^4}{z^4}, \frac{y^5}{z^5}, \frac{x}{z}, \frac{x\,y}{z^2}, \frac{x\,y^2}{z^3}, \frac{y^3\,x}{z^4}, \frac{x\,y^4}{z^5},
\frac{x^2}{z^2}, \frac{x^2\,y}{z^3}, \frac{x^2\,y^2}{z^4}, \frac{x^2\,y^3}{z^5}, \frac{x^3}{z^3}, \frac{x^3\,y}{z^4}, \frac{x^3\,y^2}{z^5}, \frac{x^4}{z^4}, \frac{x^4\,y}{z^5})$$

$$[n, k, d_\Gamma] = 64, 44, 15$$

$$C := \mathit{Goppa/primary}$$

Now encode the message and make 7 errors.

```
> a := [seq(1, i=1..C["k"])];
```

$$a := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]$$

```
> c := GoppaEncode(a, C);
```

$$c := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

```
> w := evalm(c):
> for i in [[11,13], [27,5], [39,14], [44, 5], [47,6], [61,5],[62,5]]
> do
> w[i[1]] := w[i[1]] + alpha^i[2]; od:
> print(w);
```

$$\Big[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{13}, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{5}, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{14}, 1, 1, 1, 1, 1+\alpha^{5}, 1, 1, 1+\alpha^{6}, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{5}, 1+\alpha^{5}, 1, 1\Big]$$

To decode the message with Sakata's approach, we have to initialize the decoding procedure first. (For detailed information about the computation one should increase the infolevel.)

```
> infolevel[`casa/finite`] := 0:
> H := GoppaPrepareSa(C):
```

Now we can decode the received word w using the Extended Algorithm.

```
> b := SakataDecode(w, C, H);
```

$$b := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

We obtain the message that we started with

```
> evalm(b-a);
```

$$[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

Chapter 5

Function Descriptions

5.1 Data Types

Functions that test object to be of a certain type are the following.

Function: algset

Type test for CASA's algebraic set type.

Calling Sequence:

$b :=$ type$(expr, algset)$
$b :=$ type$(expr, algset("impl"))$
$b :=$ type$(expr, algset("para"))$
$b :=$ type$(expr, algset("plac"))$
$b :=$ type$(expr, algset("proj"))$

Parameters:

expr : `expression`
any expression.

Result:

b : `boolean`
either true or false, depending on whether the given expression is an algebraic set or not.

Description:

The call type(*expr*,*algset*) yields true if *expr* is an algebraic set in any representation.
The call type(*expr*,*algset*(repr)) checks whether *expr* is an algebraic set in specified representation.

Examples:

```
> a1:=mkParaAlgSet([t^2,t^3],[t]);
```

$$a1 := \text{Parametric\_Algebraic\_Set}([t^2, t^3], [t])$$

```
> [op(a1)];
```

$$[[\_casaVariable\_t^2, \_casaVariable\_t^3], [\_casaVariable\_t], \text{"para"}, t, []]$$

```
> op(op(3,a1));
```

$$\text{"para"}$$

```
> type(a1,algset("para"));
```

$$true$$

```
> type(a1,algset("proj"));
```

$$false$$

See Also:

mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: casaAttributes

Checks whether the argument is a list of string=value pairs as required by CASA.

Calling Sequence:

$b :=$ type(x, *casaAttributes*)
$b :=$ ‘type/casaAttributes‘(x)

Parameters:

x : `anything`
Some Maple expression.

Result:

b : `bool`
true, if the argument x is a list of string=value pairs where string can only have certain values.

Description:

The function returns true if and only if the given parameter has the form list(string=anything) and the string is one of "basespace", "dimension", "genus", "groebnerBasis", "independentVariables", "neighborhoodTree", "properParametrization", "puiseuxTerms".
For any particular string the corresponding value is checked to be of correct type. See source code for more details.

Examples:

```
> type(["genus"=1],casaAttributes);
```

$$true$$

```
> type("genus"=1,casaAttributes);
```

$$false$$

See Also:

properties.

Function: casaVariable

Checks whether a given variable is a CASA internal variable.

Calling Sequence:

$b :=$ type(x, $casaVariable$)
$b :=$ ‘type/casaVariable‘(x)

Parameters:

x : `anything`
Some Maple expression.

Result:

b : `bool`
true, if the argument x is a CASA internal variable name.

Description:

The function returns true if and only if the given parameter has the form of a CASA internal variable, i.e., the first 14 characters should be `_casaVariable_`.

Examples:

```
> type(x,casaVariable);
```

$$false$$

```
> X := ‘casa/basics/internalVariable‘(x);
```

$$X := \_casaVariable\_x$$

```
> type(X,casaVariable);
```

$$true$$

See Also:

variableDifferentFrom.

Function: homogeneousPolynomial

Check for homogeneity of a polynomial in given variables.

Calling Sequence:

$b := \mathsf{type}(expr, homogeneousPolynomial(vars))$

Parameters:

$expr$: `expression`
any expression
$vars$: `list(name)`
list of variables

Result:

b : `boolean`
a boolean value.

Description:

The call $\mathsf{type}(expr, homogeneousPolynomial)$ checks whether $expr$ is a homogeneous polynomial. in the variables given by $vars$.

Examples:

```
>  type(x^2-c*y^2,homogeneousPolynomial([x,y]));
```

$$true$$

See Also:

homogenize, homogeneousForm, leadingForm.

Function: neighborhoodTree

Checks whether a given object is a neighborhood tree.

Calling Sequence:

$b :=$ ‘type/neighborhoodTree‘(l)
$b := \mathsf{type}(l, neighborhoodTree)$

Parameters:

l : `list(list)`
a list of lists

Result:

b : `boolean`
Either true or false, depending on whether the list of lists is a neighborhood tree or not.

Description:

This function checks whether a given object is a neighborhood tree or not.

A neighborhood tree is defined recursively. First of all, the empty list is a neighborhood tree. A list l of lists is a neighborhood tree if, first of al, the lists $l[i]$ in l contain six elements. The first element is a nonnegative integer; the second, third and fourth elements are coordinates of a point in a projective plane. The fifth element is a set of transformations. The sixth element $l[i][6]$ again is a neighborhood tree. And if, secondly, this definition ends; i.e. for every $1 \leq i \leq \mathsf{nops}(l)$ we have a certain k_i such that $l[i][6][6] \ldots [6] = []$ with k_i levels.

Examples:

```
> A := mkImplAlgSet([y^6-x^3+x^4+x^6],[x,y]);
```

$$A := \text{Implicit\_Algebraic\_Set}([y^6 - x^3 + x^4 + x^6],\ [x,\ y])$$

```
> G := neighbGraph(A);
```

$$G := [[3,\ 0,\ 0,\ 1,\ \{\},\ [[3,\ -1,\ 1,\ 0,\ \{x = x + y,\ y = y - x\},\ []]]$$
$$]]$$

```
> type(G,neighborhoodTree);
```

$$true$$

See Also:

neighbGraph.

Function: planecurve

Check for a plane algebraic curve or a system of plane algebraic curves.

Calling Sequence:

$b :=$ type$(expr, planecurve)$
$b :=$ type$(expr, planecurve(\text{"}system\text{"}))$

Parameters:

$expr$: `expression`
any expression

Result:

b : `boolean`
a boolean value.

Description:

The call type($expr$,$planecurve$) checks whether $expr$ defines a plane algebraic curve, i.e., $expr$ has to be an algebraic set in implicit representation represented by a single non-constant polynomial.
If the algebraic set is affine, the polynomial should be defined in the two variables specifying the algebraic set and should not contain any other indeterminates.

If the algebraic set is projective the polynomial should be defined in the three variables specifying the algebraic set and should not contain any other indeterminates. Furthermore, it should be homogeneous.
The call type(*expr*,*planecurve*("system")) checks whether *expr* is a system of plane curves. The defining polynomial of a system of plane curves can contain additional variables to the ones defining the algebraic set.

Examples:

```
> a1:=mkImplAlgSet([x^2-y^2],[x,y]);
```

$$a1 := \mathrm{Implicit\_Algebraic\_Set}([x^2 - y^2], [x, y])$$

```
> type(a1,planecurve);
```

$$true$$

```
> a2:=mkImplAlgSet([x^2-c*y^2],[x,y]);
```

$$a2 := \mathrm{Implicit\_Algebraic\_Set}([x^2 - c\,y^2], [x, y])$$

```
> type(a2,planecurve);
```

$$false$$

```
> type(a2,planecurve("system"));
```

$$true$$

See Also:

mkImplAlgSet.

Function: projPoint

Check for a valid point in projective space.

Calling Sequence:

$b :=$ type($point$, $projPoint$)

Parameters:

$point$: `list`
list of coordinates

Result:

b : `boolean`
A boolean value.

Description:

This function checks whether *point* is a valid point in a projective space, i.e. whether at least one coordinate is different from zero.

Examples:

```
> a := RootOf(x^3-2):
> type([1,0,0,a], projPoint);
```

$$true$$

```
> type([0,0,0,a^3-2], projPoint);
```

$$false$$

See Also:

pointInAlgSet.

5.2 Creator And Selector Functions for Algebraic Sets

Function: mkImplAlgSet

Creates an algebraic set given in its implicit representation.

Calling Sequence:

$A :=$ mkImplAlgSet$(repr, vars)$
$A :=$ mkImplAlgSet$(repr, vars, attr)$

Parameters:

$repr$: `list(polynom(casaCoeffType, `$vars$`))`
list of polynomials in $vars$ with coefficients of type casaCoeffType.
$vars$: `list(name)`
list of variables
$attr$: `list(equation)`
list of initial attributes

Result:

A : `algset("impl")`
A Maple object representing an algebraic set.

Description:

The function works as an input interface to the datatype algebraic set. It takes as argument a list or set of polynomials whose common zeros define the algebraic set.

Examples:

```
> A := mkImplAlgSet([x^2+y^2-4],[x,y]);
```

$$A := \mathrm{Implicit\_Algebraic\_Set}([x^2 + y^2 - 4], [x, y])$$

See Also:

mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: mkParaAlgSet

Creates an algebraic set in parametric representation

Calling Sequence:

$A :=$ mkParaAlgSet$(repr, v)$
$A :=$ mkParaAlgSet$(repr, v, attr)$

Parameters:

$repr$: `list(ratpoly(CasaCoeffType,`v`))`
list of rational functions
v : `list(name)`
internal variables
$attr$: `list(equation)`
initial attributes

Result:

A : `algset("para")`
An algebraic set in parametric representation.

Description:

The function creates an algebraic set in parametric representation. It takes as its first argument a list of rational functions which parametrize the algebraic set, whose common zeros define the algebraic set.
Its second argument is the list of variables used as indeterminates in the algebraic set.
The third, optional argument may be used to specify initial attributes of the algebraic set.

Examples:

```
> A := mkParaAlgSet([s^2/(s^2-1),(s^2-1)/(s^2+1)],[s]);
```

$$A := \mathrm{Parametric\_Algebraic\_Set}([\frac{s^2}{s^2-1}, \frac{s^2-1}{s^2+1}], [s])$$

See Also:

mkImplAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: mkPlacAlgSet

Creates an algebraic set represented by places.

Calling Sequence:

$A :=$ mkPlacAlgSet$(repr, v)$
$A :=$ mkPlacAlgSet$(repr, v, attr)$

Parameters:

repr : `procedure`
  powerseries giving the places of the algebraic set
v : `list(name)`
  variable
attr : `list(equation)`
  initial attributes

Result:

A : `algset("plac")`
  An algebraic set represented by places.

Description:

The function works as an input interface to the datatype algebraic set. It takes as argument a list of power series or polynomials. The polynomials are also interpreted as power series and automatically converted to power series. They describe the places of the algebraic set around the point which they are centered.
The second argument is a list of variables. It is given to distinguish from parameters.
The output is an algebraic set which consists of a list of lists of procedures which compute the coefficients of the represented power series up to an arbitrary order.

Examples:

```
> s := powseries[powpoly](T,T)/powseries[powpoly](1-T,T):
> s := powseries[evalpow](s):
> setPuiseuxExpansion(11);
```

$$10$$

```
> A := mkPlacAlgSet([[T,T^8+T^10,s],[s,T^3,1]],[T]);
```

$$A := \text{Places\_Algebraic\_Set}([[T,\, T^8 + T^{10}, T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + T^{10} + \mathrm{O}(T^{11})], [T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + T^{10} + \mathrm{O}(T^{11}),\, T^3,\, 1]], [T])$$

```
> A := mkPlacAlgSet([[T,T^8+T^10,s],[s,T^3,1]],[T],
> ["puiseuxTerms"=14]);
```

$$A := \text{Places\_Algebraic\_Set}([[T,\, T^8 + T^{10}, T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + T^{10} + T^{11} + T^{12} + T^{13} + \mathrm{O}(T^{14})], [T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + T^{10} + T^{11} + T^{12} + T^{13} + \mathrm{O}(T^{14}),\, T^3,\, 1]], [T])$$

See Also:

powseries, mkImplAlgSet, mkParaAlgSet, mkProjAlgSet.

Function: mkProjAlgSet

Creates an algebraic set in projected representation.

Calling Sequence:

$A :=$ mkProjAlgSet(l, v)
$A :=$ mkProjAlgSet$(l, v, attr)$

Parameters:

l : `list(list(polynom),list(ratpoly))`
A list of polynomials and a list of rational functions.
v : `list(name)`
A list of indeterminates (not including parameters).
$attr$: `list(equation)`
Some initial attributes.

Result:

A : `algset("proj")`
An algebraic set in projected representation.

Description:

The function works as an input interface to the datatype algebraic set. It takes as argument a list of two items. The first is an algebraic set, usually a hyperplane in a lower dimensional space, given in implicit form by a list of polynomials. The second item is a list of rational functions which map the given algebraic set into a higher dimensional space.
The second argument is a list of variables.
The third, optional argument may be used to specify initial attributes of the algebraic set.

Examples:

```
> A := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}($$
$$[[u^2 - v^3 + v^2], [u + v, u - v, \frac{1}{v}]], [u, v])$$

See Also:

mkImplAlgSet, mkParaAlgSet, mkPlacAlgSet.

Function: mkAlgSet

Creates an algebraic set from a given one by applying some substitutions.

Calling Sequence:

$B :=$ mkAlgSet(A, s)
$B :=$ mkAlgSet$(A, s, vars)$

Parameters:

A : `algset`
  An algebraic set in some representation.
s : `list(casaVariable=anything)`
  List of substitutions.
$vars$: `list(casaVariable)`
  A list of new main variables.

Result:

B : `algset`
  A Maple object representing an algebraic set.

Description:

The function applies the substitution s to the generators and the variable list of the algebraic set A and forms a new algebraic set from the resulting data.
If the third parameter is not given, the variable list of the algebraic set A will be taken (even if one of the variables will be replaced in the substitution s.
If A is affine/projective B will also be affine/projective.

Examples:

```
> A := mkImplAlgSet([a^2+b-4],[a,b]);
```

$$A := \text{Implicit\_Algebraic\_Set}([a^2 + b - 4], [a, b])$$

```
> mkAlgSet(A,[b=3],[a]);
```

$$\text{Implicit\_Algebraic\_Set}([a^2 - 1], [a])$$

```
> mkAlgSet(A,[b=3]);
```

$$\text{Implicit\_Algebraic\_Set}([a^2 - 1], [a, b])$$

```
> mkAlgSet(A,[b=a,a=b],[a,b]);
```

$$\text{Implicit\_Algebraic\_Set}([b^2 + a - 4], [a, b])$$

```
> B := toPara(A);
```

$$B := \text{Parametric\_Algebraic\_Set}([-x, -x^2 + 4], [x])$$

```
> x := variableList(B)[1];
```

$$x := x$$

```
> mkAlgSet(B,[x=x+d]);
```

$$\text{Parametric\_Algebraic\_Set}([-x - d, -x^2 - 2\,d\,x - d^2 + 4], [x])$$

See Also:

mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: properties

Shows every information that is already known about the algebraic set.

Calling Sequence:

$B :=$ properties(A)

Parameters:

A : `algset`
An algebraic set.

Result:

B : `algset`
The same algebraic set as A.

Description:

The function shows every single detail that is internally stored in the CASA algebraic set structure, but does not initiate any computation itself. So one might see things that CASA programmers decided to store within the structure after a heavy computation in order not to recompute it again and again, like, for example, dimension, a Gröbner basis, the genus, a neighborhood graph, etc.

Examples:

```
> A := mkImplAlgSet([x^2-y],[x,y]);
```

$$A := \text{Implicit\_Algebraic\_Set}([x^2 - y], [x, y])$$

```
> properties(A);
```

"No additional information is stored."

$$\text{Implicit\_Algebraic\_Set}([x^2 - y], [x, y])$$

```
> g := genus(A);
```

$$g := 0$$

```
> properties(A);
```

"The algebraic set is known to have the following pr \
operties:"

"It has genus 0"

"A neighborhood graph is: "

[]

$$\text{Implicit\_Algebraic\_Set}([x^2 - y], [x, y])$$

See Also:

isProjective, dimension, genus, neighbGraph, Groebnerbasis, properParametrization, setPuiseuxExpansion.

Function: generators

Returns the generators of an algebraic set.

Calling Sequence:

$r :=$ generators(A)

Parameters:

a : `algset`
An algebraic set.

Result:

r : {`list(polynom),list(ratpoly),list(list(polynom),list(ratpoly)),procedure`}
The algebraic objects representing the algebraic set A.

Description:

The function returns the algbraic objects which represent an algebraic set, i.e. for implicit representation: the list of polynomials; parametric representation: the list of rational functions; projected representation: [list of polynomials, list of rational functions]; representation by places: list of list of power series.

Examples:

```
> A := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}($$
$$[[u^2 - v^3 + v^2],\ [u + v,\ u - v,\ \frac{1}{v}]],\ [u,\ v])$$

```
> generators(A);
```

$$[[u^2 - v^3 + v^2],\ [u + v,\ u - v,\ \frac{1}{v}]]$$

See Also:

mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet, properties, variableList.

Function: parameterList

Returns the list of parameters of the given algebraic set.

Calling Sequence:

l := ‘casa/basics/parlist‘(A)

Parameters:

A : `algset`
An algebraic set

Result:

l : `list(name)`
The list of parameters of the algebraic set.

Description:

The function returns the list of parameters which is used to distinguish between indeterminates and parameters.

Examples:

```
> A := mkProjAlgSet([[a*u^2-v^3+v^2],[u+v,u-b,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}(\\ [[a\,u^2 - v^3 + v^2],\, [u+v,\, u-b,\, \frac{1}{v}]],\, [u,\, v])$$

```
> parameterList(A);
```

$$[b,\, a]$$

```
> variableList(A);
```

$$[u,\, v]$$

See Also:

variableList, mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: variableList

Returns the list of variables of the given algebraic set.

Calling Sequence:

$l :=$ variableList(A)

Parameters:

A : `algset`
An algebraic set

Result:

l : `list(name)`
The variable list of the algebraic set.

Description:

The function returns the variable list which is used to distinguish between indeterminates and parameters.

Examples:

```
> A := mkProjAlgSet([[a*u^2-v^3+v^2],[u+v,u-b,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}(\\ [[a\,u^2 - v^3 + v^2],\, [u+v,\, u-b,\, \frac{1}{v}]],\, [u,\, v])$$

```
> variableList(A);
```

$$[u,\, v]$$

```
> B := toImpl(A);
```

$$B := \text{Implicit\_Algebraic\_Set}([\\ u - a\,b^2\,z^2 - a\,v^2\,z^2 - 2\,a\,b\,v\,z^2 - v - b - 1,\\ a\,b^2\,z^3 + a\,v^2\,z^3 + 2\,a\,b\,v\,z^3 + z - 1],\, [u,\, v,\, z])$$

```
> variableList(B);
```

$$[u, v, z]$$

See Also:

mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: isProjective

Test whether the algebraic set lies in affine or projective space.

Calling Sequence:

$b :=$ isProjective(A)

Parameters:

A : `algset`
algebraic set

Result:

b : `boolean`
A boolean value which is true if the given algebraic set lies in projective space.

Description:

The function tests whether or not the given algebraic set lies in affine (false) or projective (true) space.

Examples:

```
> a := mkImplAlgSet([x^2-y],[x,y]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([x^2 - y], [x, y])$$

```
> isProjective(a);
```

$$\mathit{false}$$

```
> a := mkImplAlgSet([x^2-y*z],[x,y,z],["basespace"="projective"]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([x^2 - y\,z], [x, y, z])$$

```
> isProjective(a);
```

$$\mathit{true}$$

See Also:

mkImplAlgSet.

Function: setPuiseuxExpansion

Sets the maximum degree of the shown terms in an algebraic set represented by places.

Calling Sequence:

$r :=$ setPuiseuxExpansion($order$)

Parameters:

$order$: `posint`, *infinity*
A natural number or the value *infinity*.

Result:

r : {`posint`, *infinity*}
The order of the Puiseux expansion that was previously set.

Description:

The function sets the maximum degree of the shown terms in an algebraic set represented by places. If the parameter is *infinity*, the places will be shown in its concrete internal representation, which is procedures that generate the coefficients of the Puiseux expansion.

Examples:

```
> n := setPuiseuxExpansion(10);
```

$$n := 10$$

```
> s:=powseries[evalpow](powseries[powpoly](T,T)/powseries[powpoly](1-T,T)):
> B:=mkPlacAlgSet([[T,T^8+T^10,s],[s,T^3,1]],[T]);
```

$$B := \text{Places\_Algebraic\_Set}([[T,\ T^8 + \mathrm{O}(T^{10}),\\ T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + \mathrm{O}(T^{10})], [\\ T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + \mathrm{O}(T^{10}),\\ T^3,\ 1]],\ [T])$$

```
> setPuiseuxExpansion(n);
```

$$10$$

```
> B;
```

$$\text{Places\_Algebraic\_Set}([[T,\ T^8 + \mathrm{O}(T^{10}),\\ T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + \mathrm{O}(T^{10})], [\\ T + T^2 + T^3 + T^4 + T^5 + T^6 + T^7 + T^8 + T^9 + \mathrm{O}(T^{10}),\\ T^3,\ 1]],\ [T])$$

See Also:

mkPlacAlgSet, properties.

5.3 Conversion Between Different Representations

Function: toImpl

Converts an algebraic set given in parametric or projected representation to an algebraic set in implicit representation in affine space.

Calling Sequence:

$B :=$ toImpl(A)
$B :=$ toImpl$(A, vars)$

Parameters:

A : {`algset("para"),algset("proj"),algset("impl")`}
algebraic set in parametric implicit or projected representation
$vars$: `list(name)`
List of variables.

Result:

B : `algset("impl")`
The implicit representation of the algebraic set A.

Description:

This function takes an algebraic set and converts it into an implicit representation in affine space. The dimension of the target space can be prescribed by the number of variables that can be given as a second argument.
If no variables are given, the function tries to take the variables from the given algebraic set and add some more if necessary to embed the algebraic set into an affine space of lowest dimension.

Examples:

```
> A := mkParaAlgSet([t,t^2,t^3],[t]);
```

$$A := \text{Parametric\_Algebraic\_Set}([t,\, t^2,\, t^3],\, [t])$$

```
> toImpl(A,[x,y,z]);
```

$$\text{Implicit\_Algebraic\_Set}(\\ [x^2 - y,\, -z + x\,y,\, -y^2 + z\,x,\, -z^2 + y^3],\, [x,\, y,\, z])$$

```
> A := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}(\\ [[u^2 - v^3 + v^2],\, [u + v,\, u - v,\, \frac{1}{v}]],\, [u,\, v])$$

```
> toImpl(A);
```

$$\text{Implicit\_Algebraic\_Set}(\\ [u - 2\,v^2\,z^2 - 4\,z\,v - v - 4,\, -1 + v^2\,z^3 + 2\,v\,z^2 + 2\,z], \\ [u,\, v,\, z])$$

```
> toImpl(A,[a,b,c]);
```

$$\text{Implicit\_Algebraic\_Set}(\\ [a - 2\,b^2\,c^2 - 4\,c\,b - b - 4,\, -1 + b^2\,c^3 + 2\,b\,c^2 + 2\,c] \\ ,\, [a,\, b,\, c])$$

See Also:

toPara, toProj, toPlac, toAffine, toProjective.

Function: toPara

Converts an algebraic set given in implicit or projected representation to an algebraic set in parametric representation.

Calling Sequence:

$B :=$ toPara(A)
$B :=$ toPara(A, var)
$B :=$ toPara$(A, var, options)$

Parameters:

A : `algset("impl"),algset("proj"),algset("para")`
Algebraic set in impl para or projected representation.

var : `name`
A variable name that is used for the parametrization.

$options$: `list`
A list of options. Where an option is either one of the strings "optimal" (compute a parametrization over an optimal extension field), "realpar" (compute a real parametrization), "check" (plug in the resulting parametrization in the given equations), "points"= $\langle$list of simple points on the curve$\rangle$ (a list of points on the curve that could be helpful for the parametrization. The points which are not on the curve will be automatically removed.
Note, the user should know that supplying simple points may not be a good idea for a non-expert since it may lead to a non-optimal parametrization.

Result:

B : `algset("para")`
A parametric representation of the algebraic set A.

Description:

This function takes an algebraic set and converts it into a parametric representation in affine space. Currently, it may be applied only to one-dimensional algebraic sets (i.e. both plane and space algebraic curves).
The variables $vars$ will be taken for the parametrization. If no variables are given, the function tries to take the variables from the given algebraic set.

Examples:

```
> A:=mkImplAlgSet([y^2*z^3-x^5],[x,y,z],["basespace"="projective"]);
```

$$A := \text{Implicit\_Algebraic\_Set}([y^2\, z^3 - x^5],\, [x,\, y,\, z])$$

```
> toPara(A);
```

$$\text{Parametric\_Algebraic\_Set}([u^2,\, -u^5,\, 1],\, [u])$$

```
> A:=mkImplAlgSet([y^4-10*x*y^3+35*x^2*y^2-50*x^3*y+24*x^4+x^3],[x,y]);
```

$$A := \text{Implicit\_Algebraic\_Set}(\\ [y^4 - 10\, x\, y^3 + 35\, x^2\, y^2 - 50\, x^3\, y + 24\, x^4 + x^3],\, [x,\, y])$$

```
> toPara(A);
```

$$\begin{aligned}&\mathrm{Parametric\_Algebraic\_Set}([(-584\,\%2+292\,z-4\,\%3\\&+148\,z^4+444\,\%1-432\,z^2-37\\&-36\,\mathrm{RootOf}(\_Z^2+1)-8\,z^3\\&+144\,\mathrm{RootOf}(\_Z^2+1)\,z^4)/(1896\,\%3+713\,z^2+888\\&+6252\,z^3+3346\,\%2-5158\,z\\&+1300\,\mathrm{RootOf}(\_Z^2+1)\,z^4-1625\,z^4\\&+864\,\mathrm{RootOf}(\_Z^2+1)-9246\,\%1),z(-222\,\%1\\&+37+36\,\mathrm{RootOf}(\_Z^2+1)+146\,\%2-219\,z\\&+3\,\%3+216\,z^2+2\,z^3)/(1896\,\%3+713\,z^2+888\\&+6252\,z^3+3346\,\%2-5158\,z\\&+1300\,\mathrm{RootOf}(\_Z^2+1)\,z^4-1625\,z^4\\&+864\,\mathrm{RootOf}(\_Z^2+1)-9246\,\%1)],\,[z])\\&\%1:=\mathrm{RootOf}(\_Z^2+1)\,z^2\\&\%2:=\mathrm{RootOf}(\_Z^2+1)\,z^3\\&\%3:=\mathrm{RootOf}(\_Z^2+1)\,z\end{aligned}$$

```
> toPara(A,t,["optimal"]);
```

$$\mathrm{Parametric\_Algebraic\_Set}([-\frac{-48384\,t^3+20736\,t^4+42336\,t^2-16464\,t+2401}{482699\,t^2-275086\,t+101065\,t^4-366926\,t^3+57624}$$
$$,\,-7\,\frac{t\,(1728\,t^3-343-3024\,t^2+1764\,t)}{482699\,t^2-275086\,t+101065\,t^4-366926\,t^3+57624}$$
$$],\,[t])$$

```
> A:=mkImplAlgSet([x^2-y^3,z+y+x],[x,y,z]);
```

$$A:=\mathrm{Implicit\_Algebraic\_Set}([x^2-y^3,\,z+y+x],\,[x,\,y,\,z])$$

```
> toPara(A);
```

$$\begin{aligned}&\mathrm{Parametric\_Algebraic\_Set}([-\frac{3\,y+y^3+3\,y^2+1}{3\,y-3\,y^2-1+y^3},\\&\frac{1+y^2+2\,y}{y^2-2\,y+1},\,2\,\frac{1+y^2+2\,y}{3\,y-3\,y^2-1+y^3}],\,[y])\end{aligned}$$

```
> A := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$\begin{aligned}&A:=\mathrm{Projected\_Algebraic\_Set}(\\&[[u^2-v^3+v^2],\,[u+v,\,u-v,\,\frac{1}{v}]],\,[u,\,v])\end{aligned}$$

```
> toPara(A);
```

$$\begin{aligned}&\mathrm{Parametric\_Algebraic\_Set}(\\&[\frac{-1+x-x^2+x^3}{x^3},\,-\frac{1+x+x^2+x^3}{x^3},\,\frac{x^2}{1+x^2}],\,[x])\end{aligned}$$

See Also:

toImpl, toProj, toPlac.

Function: toPlac

Converts an algebraic set given in implicit parametric or projected representation to an algebraic set represented by places.

Calling Sequence:

$B :=$ toPlac(A)
$B :=$ toPlac(A, p)
$B :=$ toPlac(A, p, var)

Parameters:

A : {`algset("impl"),algset("para"),algset("proj"),algset("plac")`}
algebraic set in implicit, parametric, place or projected representation

p : {`casaCoeffType,list(casaCoeffType)`}
If p is a list, it specifies the center of the expansion. Otherwise such a point is computed by taking the point on the curve A that corresponts to the parameter p.

var : `list(name)`
A variable to be used in the Puiseux expansion.

Result:

B : `algset("plac")`
The representation of the algebraic set given by places.

Description:

This function takes an algebraic set and converts it into an representation given by places. If the center point is omitted, it defaults to the origin (which must be on the curve).
If no variable is given, the function chooses one.

Examples:

```
> A := mkParaAlgSet([t,t^2,t^3],[t]);
```

$$A := \text{Parametric\_Algebraic\_Set}([t,\, t^2,\, t^3],\, [t])$$

```
> toPlac(A,[0,0,0],t);
```

$$\text{Places\_Algebraic\_Set}([[t + \mathrm{O}(t^{11}),\, t^2 + \mathrm{O}(t^{11}),\, t^3 + \mathrm{O}(t^{11})]],\, [t])$$

```
> A := mkProjAlgSet([[u^2-v^3+v^2],[u+v,u-v,1/v]],[u,v]);
```

$$A := \text{Projected\_Algebraic\_Set}([[u^2 - v^3 + v^2],\, [u+v,\, u-v,\, \frac{1}{v}]],\, [u,\, v])$$

```
> toPlac(A,[1,-1,1]);
```

$$\text{Places\_Algebraic\_Set}([],\, [u])$$

See Also:

toImpl, toPara, toProj, toAffine, toProjective.

Function: toProj

Converts an algebraic set given in parametric or implicit representation to an algebraic set in projected representation.

Calling Sequence:

$B :=$ toProj(A)
$B :=$ toProj$(A, vars)$

Parameters:

A : {`algset("impl"),algset("para"),algset("proj")`}
Algebraic set in implicit projected or parametric representation.
$vars$: `list(name)`
List of variables.

Result:

B : `algset("proj")`
A projected representation of the algebraic set A.

Description:

This function takes an algebraic set and converts it into a projected representation in affine space.
The variables $vars$ will be taken for the representation in projected form. There must be at least $d+1$ variables where d is the dimension of the given algebraic set.

Examples:

```
> a:=mkImplAlgSet([y^2*z^3-x^5],[x,y,z],["basespace"="projective"]);
```

$$a := \text{Implicit\_Algebraic\_Set}([y^2\, z^3 - x^5],\, [x,\, y,\, z])$$

```
> toProj(a);
```

$$\text{Projected\_Algebraic\_Set}([[-y^2\, z^3 + x^5],\, [x,\, y,\, z]],\, [x,\, y,\, z])$$

```
> a:=mkImplAlgSet([y^4-10*x*y^3+35*x^2*y^2-50*x^3*y+24*x^4+x^3],[x,y]);
```

$$a := \text{Implicit\_Algebraic\_Set}(\\ [y^4 - 10\, x\, y^3 + 35\, x^2\, y^2 - 50\, x^3\, y + 24\, x^4 + x^3],\, [x,\, y])$$

```
> b:=toPara(a);
```

$$b := \text{Parametric_Algebraic_Set}([16z^4\,(-15 + 473\,\%2)/(
-170008\,\text{RootOf}(_Z^2+1)\,z^2\,\%1 - 760920\,z^2
- 111360\,z - 71232\,z^3\,\text{RootOf}(_Z^2+1)\,\%1
- 1473600\,z^3 - 7568\,\%2
+ 87135\,z^4\,\text{RootOf}(_Z^2+1)\,\%1 - 737145\,z^4 + 240
- 68352\,\text{RootOf}(_Z^2+1)\,z\,\%1), 8z^3(
-3555\,z + 30 - 946\,\%2 + 229\,\text{RootOf}(_Z^2+1)\,z\,\%1
)/(-170008\,\text{RootOf}(_Z^2+1)\,z^2\,\%1 - 760920\,z^2
- 111360\,z - 71232\,z^3\,\text{RootOf}(_Z^2+1)\,\%1
- 1473600\,z^3 - 7568\,\%2
+ 87135\,z^4\,\text{RootOf}(_Z^2+1)\,\%1 - 737145\,z^4 + 240
- 68352\,\text{RootOf}(_Z^2+1)\,z\,\%1)],\,[z])$$

$$\%1 := \text{RootOf}(\_Z^2 - 15)$$

$$\%2 := \text{RootOf}(\_Z^2 + 1)\,\%1$$

```
> toProj(b,[u,v]);
```

$$\text{Projected_Algebraic_Set}([[v], [16u^4\,(-15 + 473\,\%2)/(
-170008\,\text{RootOf}(_Z^2+1)\,u^2\,\%1 - 760920\,u^2
- 111360\,u - 71232\,u^3\,\text{RootOf}(_Z^2+1)\,\%1
- 1473600\,u^3 - 7568\,\%2
+ 87135\,u^4\,\text{RootOf}(_Z^2+1)\,\%1 - 737145\,u^4 + 240
- 68352\,\text{RootOf}(_Z^2+1)\,u\,\%1), 8u^3(-3555\,u
+ 30 - 946\,\%2 + 229\,\text{RootOf}(_Z^2+1)\,u\,\%1)/(
-170008\,\text{RootOf}(_Z^2+1)\,u^2\,\%1 - 760920\,u^2
- 111360\,u - 71232\,u^3\,\text{RootOf}(_Z^2+1)\,\%1
- 1473600\,u^3 - 7568\,\%2
+ 87135\,u^4\,\text{RootOf}(_Z^2+1)\,\%1 - 737145\,u^4 + 240
- 68352\,\text{RootOf}(_Z^2+1)\,u\,\%1)]],\,[u,\,v])$$

$$\%1 := \text{RootOf}(\_Z^2 - 15)$$

$$\%2 := \text{RootOf}(\_Z^2 + 1)\,\%1$$

```
> toProj(a,[u,v]);
```

$$\text{Projected_Algebraic_Set}([
[-10\,u\,v^3 + v^4 + 24\,u^4 + 35\,u^2\,v^2 + (-50\,v + 1)\,u^3],
[u,\,v]],\,[u,\,v])$$

See Also:

toImpl, toPara, toPlac.

Function: toAffine

Converts the algebraic set to the corresponding algebraic set in affine space.

Calling Sequence:

$A :=$ toAffine(a)
$A :=$ toAffine(a, x)

Parameters:

a : `algset("impl")`
Algebraic set in affine or projective implicit representation.

x : `name`
The variable with respect to which the dehomogenization of the defining polynomials will be done. This variable must be among the variables in the variable list of a.

Result:

A : `algset("impl")`
The corresponding algebraic set in affine space.

Description:

The function converts an algebraic set in implicit representation to the corresponding algebraic set in implicit form but lying in affine space.
If the dehomogenization variable is not provided, the last variable of the variable list of a will be taken.
If the given algebraic set is already in affines space, it is returned immediately.

Examples:

```
> a:=mkImplAlgSet([x^2*z-2*y^3+4*x*y^2],[x,y,z],
> ["basespace"="projective"]);
```

$$a := \text{Implicit\_Algebraic\_Set}([x^2\,z - 2\,y^3 + 4\,x\,y^2],\,[x,\,y,\,z])$$

```
> b:=toAffine(a,x);
```

$$b := \text{Implicit\_Algebraic\_Set}([z - 2\,y^3 + 4\,y^2],\,[y,\,z])$$

```
> c:=toAffine(a,y);
```

$$c := \text{Implicit\_Algebraic\_Set}([x^2\,z - 2 + 4\,x],\,[x,\,z])$$

```
> [isProjective(a),isProjective(b),isProjective(c)];
```

$$[\mathit{true},\,\mathit{false},\,\mathit{false}]$$

See Also:

variableList, toProjective.

Function: toProjective

Converts the algebraic set to the corresponding algebraic set in projective space.

Calling Sequence:

$P :=$ toProjective(a)
$P :=$ toProjective(a, x)

Parameters:

a : `algset("impl")`
Algebraic set in affine or projective implicit representation.

x : `name`
The homogenization variable. This variable must not be in the variable list of a.

Result:

P : `algset("impl")`
The corresponding algebraic set in projective space.

Description:

The function converts an algebraic set in implicit representation to the corresponding algebraic set in implicit form but lying in projective space.
If the given algebraic set is already in projective space, it is returned immediately.
If no homogenization variable is provided, a new variable will be chosen that is neither among the variables nor the parameters of a.

Examples:

```
> a:=mkImplAlgSet([x^2-2*y^3+4*x*y^2],[x,y]);
```

$$a := \text{Implicit\_Algebraic\_Set}([x^2 - 2\,y^3 + 4\,x\,y^2],\ [x,\ y])$$

```
> b:=toProjective(a);
```

$$b := \text{Implicit\_Algebraic\_Set}([x^2\,z - 2\,y^3 + 4\,x\,y^2],\ [x,\ y,\ z])$$

```
> c:=toProjective(a,H);
```

$$c := \text{Implicit\_Algebraic\_Set}([H\,x^2 - 2\,y^3 + 4\,x\,y^2],\ [x,\ y,\ H])$$

```
> [isProjective(a),isProjective(b),isProjective(c)];
```

$$[\mathit{false},\ \mathit{true},\ \mathit{true}]$$

See Also:

variableList, toAffine, isProjective.

5.4 Operations on Algebraic Sets

Function: dimension

The function Computes the dimension of an algebraic set.

Calling Sequence:

$d :=$ dimension(a)

Parameters:

a : `algset`

An algebraic set.

Result:

d : `int`
The dimension of the algebraic set a.

Description:

The function computes the dimension of an algebraic set.

Examples:

```
> a := mkImplAlgSet([x^2-3*x*y^2-7*y^3,x^2-z^2-1],[x,y,z]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([x^2 - 3\,x\,y^2 - 7\,y^3,\ x^2 - z^2 - 1],\ [x,\ y,\ z])$$

```
> dimension(a);
```

$$1$$

See Also:

independentVariables, mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: decompose

Decomposition of an algebraic set into irreducible components.

Calling Sequence:

$s :=$ decompose(A)
$s :=$ decompose(A, *absolute*)

Parameters:

A : `algset`
An algebraic set
absolute :
The name 'absolute'.

Result:

s : `exprseq`
The decomposition of an algebraic set into its irreducible components.

Description:

The function decompose computes the decomposition of an algebraic set into its irreducible components.
On default the decomposition is done over the rational numbers. The optional argument *absolute* only works for algebraic sets in projected form and performs the decomposition over the algebraic closure.

Examples:

```
> a1 := mkImplAlgSet([x^2+y^2-4,x^2+z^2-4],[x,y,z]);
```

$a1 :=$ Implicit_Algebraic_Set$([x^2 + y^2 - 4,\ x^2 + z^2 - 4],$
$[x,\ y,\ z])$

```
> [decompose(a1)];
```

[Implicit_Algebraic_Set$([x^2 + y^2 - 4,\ z + y],\ [x,\ y,\ z])$,
Implicit_Algebraic_Set$([x^2 + y^2 - 4,\ -z + y],\ [x,\ y,\ z])$
]

```
> a2 := mkProjAlgSet([[u^2-2*v^2],[u,v,u-v]],[u,v]);
```

$a2 :=$ Projected_Algebraic_Set$([[u^2 - 2\,v^2],\ [u,\ v,\ u - v]],$
$[u,\ v])$

```
> [decompose(a2,"absolute")];
```

[Projected_Algebraic_Set(
$[[v - \frac{1}{2}\,\mathrm{RootOf}(\_Z^2 - 2)\,u],\ [u,\ v,\ u - v]],\ [u,\ v])$,
Projected_Algebraic_Set(
$[[v + \frac{1}{2}\,\mathrm{RootOf}(\_Z^2 - 2)\,u],\ [u,\ v,\ u - v]],\ [u,\ v])$]

See Also:

computeRadical, mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: singLocus

Compute singularities of an algebraic set.

Calling Sequence:

$B :=$ singLocus(A)
$B :=$ singLocus$(A, vars)$

Parameters:

A : `algset`
An algebraic set.
$vars$: `list(name)`
A list of CASA variables in which the resulting algebraic set will be specified.

Result:

B : `algset("impl")`
The algebraic set of singularities of the given algebraic set.

Description:

The function computes the algebraic set of all singularities of a given algebraic set

This can only be done for algebraic sets not given by places. For algebraic sets in implicit form the partial derivatives with respect to the variables in variableList(A) are computed. Algebraic sets in parametric or in projected form are first converted to implicit form.
A second parameter can be given to specify the variables that should be used for the resulting algebraic set.
The function returns an algebraic set in implicit form in affine space.

Examples:

```
> A := mkImplAlgSet([x^2+x^3-y^2],[x,y]);
```

$$A := \mathrm{Implicit\_Algebraic\_Set}([x^2 + x^3 - y^2], [x, y])$$

```
> L := singLocus(A);
```

$$L := \mathrm{Implicit\_Algebraic\_Set}([-2\,y,\, x^2 + x^3 - y^2,\, 2\,x + 3\,x^2], [x, y])$$

```
> tsolve(L);
```

$$\mathrm{Implicit\_Algebraic\_Set}([y, -2\,x], [x, y])$$

See Also:

singularities, tsolve.

Function: setRandomParameters

Choose undetermined parameters at random.

Calling Sequence:

$B :=$ setRandomParameters(A)
$B :=$ setRandomParameters(A, rng)

Parameters:

A : `algset`
system of algebraic sets
rng : `integer..integer`
Range in which the parameters will be chosen.

Result:

B : `algset`
A single algebraic set.

Description:

The function returns a single algebraic set from the system of sets A containing parameters. The result is obtained by setting undetermined parameters to random values in rng (default is $-10 \ldots 10$). If an unevaluated name is specified, it is assigned the set of selected values - The standard Maple function rand is used to select random values.

Examples:

```
> a:=passGenCurve(5,[[[0,1,0],2],[[0,0,1],3],[[1,1,-1],2]],u,[x,y,z]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([(-u_{[4,\,1,\,0]} - u_{[3,\,2,\,0]} - u_{[2,\,3,\,0]} + u_{[2,\,1,\,2]} + u_{[0,\,3,\,2]} + u_{[3,\,0,\,2]} + u_{[1,\,2,\,2]})x^5 + u_{[4,\,1,\,0]}\,x^4\,y + u_{[3,\,2,\,0]}\,x^3\,y^2 + u_{[2,\,3,\,0]}\,x^2\,y^3 + (-u_{[4,\,1,\,0]} - 2\,u_{[3,\,2,\,0]} - 3\,u_{[2,\,3,\,0]} + u_{[2,\,2,\,1]} + 2\,u_{[1,\,3,\,1]} + u_{[2,\,1,\,2]} - u_{[0,\,3,\,2]} + 2\,u_{[3,\,0,\,2]})z\,x^4 + (u_{[4,\,1,\,0]} + 2\,u_{[3,\,2,\,0]} + 3\,u_{[2,\,3,\,0]} - 2\,u_{[2,\,2,\,1]} - 3\,u_{[1,\,3,\,1]} + u_{[2,\,1,\,2]} + 2\,u_{[1,\,2,\,2]} + 3\,u_{[0,\,3,\,2]})x^3\,y\,z + u_{[2,\,2,\,1]}\,x^2\,y^2\,z + u_{[1,\,3,\,1]}\,y^3\,z\,x + u_{[3,\,0,\,2]}\,z^2\,x^3 + u_{[2,\,1,\,2]}\,x^2\,y\,z^2 + u_{[1,\,2,\,2]}\,x\,y^2\,z^2 + u_{[0,\,3,\,2]}\,y^3\,z^2],\ [x,\,y,\,z])$$

```
> setRandomParameters(a);
```

$$\mathrm{Implicit\_Algebraic\_Set}([7\,x^5 - y\,x^4 + 4\,x^3\,y^2 - 5\,y^3\,x^2 + 24\,z\,x^4 - 15\,x^3\,y\,z + 3\,x^2\,y^2\,z - 2\,y^3\,z\,x + 4\,z^2\,x^3 + 10\,x\,y^2\,z^2 - 9\,y^3\,z^2],\ [x,\,y,\,z])$$

See Also:

passGenCurve.

Function: pointInAlgSet

Tests whether a point is in a given algebraic set.

Calling Sequence:

$b :=$ pointInAlgSet($point, A$)

Parameters:

point : `list`
An point.

A : `algset`
An algebraic set that is in implicit representation or can be converted to implicit representation.

Result:

b : `boolean`
A boolean value indicating whether the given point is in the given algebraic set.

Description:

This procedure tests whether *point* (given as a list of coordinates) lies in the algebraic set A.
If necessary, the algebraic set is converted to implicit representation.

Examples:

```
> a1:=mkImplAlgSet([x^2+y,z^3+x*z-1],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^2 + y,\ z^3 + x\,z - 1],\ [x,\ y,\ z])$$

```
> pointInAlgSet([0,0,1],a1);
```

$$true$$

```
> a2:=mkProjAlgSet([[z^3+x*z-1],[x,-(-2*z^3+1+z^6)/z^2, z]],[x, z]);
```

$$a2 := \text{Projected\_Algebraic\_Set}($$
$$[[z^3 + x\,z - 1],\ [x,\ -\frac{-2\,z^3 + 1 + z^6}{z^2},\ z]],\ [x,\ z])$$

```
> pointInAlgSet([0,0,1],a2);
```

$$true$$

See Also:

toImpl, toProjective.

5.5 Operations on Algebraic Sets in Implicit Form

Function: Groebnerbasis

Compute a Gröbner basis for an implicitly given algebraic set.

Calling Sequence:

$g :=$ Groebnerbasis(A)
$g :=$ Groebnerbasis$(A, order)$

Parameters:

A : `algset("impl")`
algebraic set in implicit representation,
$order$: `name`
The name 'tdeg' or 'plex'.

Result:

g : `list(polynom)`
Gröbner basis for the given algebraic set.

Description:

The function computes a Gröbner basis for an implicitly given algebraic set. The ordering of the variables is according to the list of variables, the greatest variables comes first.
The term ordering is the total degree term ordering unless it is specified differently by an optional parameter.
This function is nothing but an interface between CASA's datatype algset and Maple's package for Gröbner bases computation.

Examples:

```
> a := mkImplAlgSet([x*y*z^2-4,x^2+y^2+z^2-1,x-y-1],[x,y,z]);
```

$$a := \text{Implicit\_Algebraic\_Set}(\\ [x\,y\,z^2 - 4,\ x^2 + y^2 + z^2 - 1,\ x - y - 1],\ [x,\ y,\ z])$$

```
> Groebnerbasis(a);
```

$$[8 + z^4,\ 2\,y^2 + z^2 + 2\,y,\ x - y - 1]$$

See Also:

GWalk, mgbasis.

Function: independentVariables

Computes a set of independent variables.

Calling Sequence:

$s :=$ independentVariables(A)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.

Result:

s : `set(name)`
a maximal set of independent variables of the algebraic set

Description:

The function computes a maximal set of independent variables for an algebraic set in implicit form. The dimension of the algebraic set is exactly the number of independent variables.

Examples:

```
> a := mkImplAlgSet([x^2-3*x*y^2-7*y^3, x^2-z^2-1],[x,y,z]);
```

$$a := \text{Implicit\_Algebraic\_Set}([x^2 - 3\,x\,y^2 - 7\,y^3,\ x^2 - z^2 - 1],\\ [x,\ y,\ z])$$

```
> independentVariables(a);
```

$$\{z\}$$

See Also:

dimension, mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet.

Function: equalBaseSpaces

Test whether two algebraic sets in implicit representation live in the same space.

Calling Sequence:

$b :=$ equalBaseSpaces(A, B)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.

B : `algset("impl")`
An algebraic set in implicit representation.

Result:

b : `boolean`
A boolean value indicating whether the algebraic sets A and B have the same variables and lie both in affine or projective space.

Description:

The function tests whether two algebraic sets in implicit representation live in the same space.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^3 + x^2\,y - x,\, z],\, [x,\, y,\, z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x,\, y^2 + z^2 - 1],\, [x,\, y,\, z])$$

```
> a3 := mkImplAlgSet([y^2+z^2-1],[y,z]);
```

$$a3 := \text{Implicit\_Algebraic\_Set}([y^2 + z^2 - 1],\, [y,\, z])$$

```
> equalBaseSpaces(a1,a2);
```

$$\mathit{true}$$

```
> equalBaseSpaces(a1,a3);
```

$$\mathit{false}$$

```
> equalBaseSpaces(a1,toProjective(a2));
```

$$\mathit{false}$$

See Also:

computeRadical, implUnion, implUnionLCM, implIntersect, implEqual, implEmpty, implIdealQuo.

Function: implDifference

Computes the Zariski closure of the difference of two algebraic sets.

Calling Sequence:

$C :=$ implDifference(A, B)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.
B : `algset("impl")`
An algebraic set in implicit representation.

Result:

C : `algset("impl")`
The Zariski closure of the difference of the two given algebraic sets.

Description:

The function computes the algebraic set of the Zariski closure of the difference of the given algebraic sets.
The computation is reduced to the computation of the intersection of ideals. Let $\{q_1, \ldots, q_n\}$ be a basis for $I(B)$, the ideal of B. It holds that

$$I(A) : I(B) = \bigcap_{i=1}^{n} I(A) : \langle q_i \rangle$$

where $\langle q_i \rangle$ is the principle ideal generated by q_i. Each $I(A) : \langle q_i \rangle$ can be computed again by ideal intersection. Let $\{p_1, \ldots, p_m\}$ be a basis for $I(A) \cap \langle q_i \rangle$ then $\{p_1/q_i, ..., p_m/q_i\}$ is a basis for $I(A) : \langle q_i \rangle$.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^3 + x^2\, y - x,\ z],\ [x,\ y,\ z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x,\ y^2 + z^2 - 1],\ [x,\ y,\ z])$$

```
> implDifference(a1,a2);
```

$$\text{Implicit\_Algebraic\_Set}([x^3 + x^2\, y - x,\ z],\ [x,\ y,\ z])$$

```
> implDifference(a1,a1);
```

$$\text{Implicit\_Algebraic\_Set}([1],\ [x,\ y,\ z])$$

See Also:

implUnion, implUnionLCM, implIntersect, implEqual, implEmpty, implSubSet, implIdealQuo.

Function: implEmpty

Tests whether an algebraic set is empty.

Calling Sequence:

$b :=$ implEmpty(A)

Parameters:

A : `algset("impl")`
Algebraic set in implicit representation.

Result:

b : `boolean`
A boolean value indicating whether the given algebraic set is empty or not.

Description:

The function tests if an algebraic set in implicit form is empty. It is tested whether '1' is in the Groebner basis for the ideal of A.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,y^2+z^2-1,x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}(\\ [x^3 + x^2\,y - x,\ y^2 + z^2 - 1,\ x,\ z],\ [x,\ y,\ z])$$

```
> implEmpty(a1);
```

$$\mathit{false}$$

```
> a2 := mkImplAlgSet([x^2-1,x^2-3],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x^2 - 1,\ x^2 - 3],\ [x,\ y,\ z])$$

```
> implEmpty(a2);
```

$$\mathit{true}$$

See Also:

implUnion, implUnionLCM, implIntersect, implSubSet, implEqual, implIdealQuo.

Function: implEqual

Test whether two algebraic sets in implicit representation are equal.

Calling Sequence:

$b :=$ implEqual(A, B)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.
B : `algset("impl")`
An algebraic set in implicit representation.

Result:

b : `boolean`
A boolean value indicating whether the two given algebraic sets are equal.

Description:

The function implEqual tests if two algebraic sets in implicit form are equal. There is implEqual(A,B) if and only if (implSubSet(A,B) and implSubSet(B,A)).

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,y^2+z^2-1,x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}(\\ [x^3 + x^2\,y - x,\ y^2 + z^2 - 1,\ x,\ z],\ [x,\ y,\ z])$$

```
> a2 := tsolve(a1);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([z,\ y^2 - 1,\ x],\ [x,\ y,\ z])$$

```
> implEqual(a1,a2);
```

$$true$$

See Also:

implUnion, implUnionLCM, implIntersect, implSubSet, implEmpty, implIdealQuo.

Function: implIdealQuo

Computes the Zariski closure of the difference of two algebraic sets.

Calling Sequence:

$C :=$ implIdealQuo(A, B)

Parameters:

A : `algset("impl")`
an algebraic set in implicit representation
B : `algset("impl")`
an algebraic set in implicit representation

Result:

C : `algset("impl")`
the Zariski closure of the difference of the two given algebraic sets.

Description:

The function computes the algebraic set of the ideal quotient of the ideals of the given algebraic sets.
The computation is reduced to the computation of the intersection of ideals. Let $\{q_1, \ldots, q_n\}$ be a basis for $I(B)$, the ideal of B. It holds that

$$I(A) : I(B) = \bigcap_{i=1}^{n} I(A) : \langle q_i \rangle$$

where $\langle q_i \rangle$ is the principle ideal generated by q_i. Each $I(A) : \langle q_i \rangle$ can be computed again by ideal intersection. Let $\{p_1, \ldots, p_m\}$ be a basis for $I(A) \cap \langle q_i \rangle$ then $\{p_1/q_i, ..., p_m/q_i\}$ is a basis for $I(A) : \langle q_i \rangle$.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^3 + x^2\,y - x,\ z],\ [x,\ y,\ z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x,\ y^2 + z^2 - 1],\ [x,\ y,\ z])$$

```
> implIdealQuo(a1,a2);
```

$$\text{Implicit\_Algebraic\_Set}([x^3 + x^2\,y - x,\ z],\ [x,\ y,\ z])$$

```
> implIdealQuo(a1,a1);
```

$$\text{Implicit\_Algebraic\_Set}([1],\ [x,\ y,\ z])$$

See Also:

implUnion, implUnionLCM, implIntersect, implEqual, implEmpty, implSubSet.

Function: implIntersect

Computes the intersection of algebraic sets.

Calling Sequence:

$I :=$ implIntersect(As)

Parameters:

As : `exprseq(algset("impl"))`
algebraic sets in implicit representation

Result:

I : `algset("impl")`
the intersection of the given sequence of algebraic sets.

Description:

The function computes the intersection of algebraic sets in implicit form by computing the union of the corresponding ideals. These ideals are given by a finite basis. A basis of the union is obtained by putting all basis elements into one new basis.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^3 + x^2\,y - x,\ z],\ [x,\ y,\ z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x,\ y^2 + z^2 - 1],\ [x,\ y,\ z])$$

```
> a3 := implIntersect(a1,a2);
```

$$a3 := \text{Implicit\_Algebraic\_Set}(\\ [x,\ z,\ x^3 + x^2\,y - x,\ y^2 + z^2 - 1],\ [x,\ y,\ z])$$

```
> tsolve(a3);
```

$$\text{Implicit\_Algebraic\_Set}([z,\ y^2 - 1,\ x],\ [x,\ y,\ z])$$

See Also:

implUnion, implUnionLCM, implSubSet, implEqual, implEmpty, implIdealQuo.

Function: implSubSet

Test whether an algebraic set is contained in another one.

Calling Sequence:

$b :=$ implSubSet(A, B)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.
B : `algset("impl")`
An algebraic set in implicit representation.

Result:

b : `boolean`
A boolean value indicating whether the algebraic set A is a subset of the algebraic set B.

Description:

The function tests if an algebraic set is a subset of another algebraic set in implicit form.
An algebraic set A is a subset of B if and only if $\mathrm{rad}(I(B))$ is a subset of $\mathrm{rad}(I(A))$, where $I(x)$ denotes the ideal of the algebraic set x and $\mathrm{rad}(I)$ denotes the radical ideal of I.

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \mathrm{Implicit\_Algebraic\_Set}([x^3 + x^2\,y - x,\, z],\, [x,\, y,\, z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \mathrm{Implicit\_Algebraic\_Set}([x,\, y^2 + z^2 - 1],\, [x,\, y,\, z])$$

```
> a3 := implIntersect(a1,a2);
```

$$a3 := \mathrm{Implicit\_Algebraic\_Set}($$
$$[x,\, z,\, x^3 + x^2\,y - x,\, y^2 + z^2 - 1],\, [x,\, y,\, z])$$

```
> implSubSet(a3,a1);
```

$$true$$

```
> implSubSet(a2,a3);
```

$$false$$

See Also:

equalBaseSpaces, computeRadical, implUnion, implUnionLCM, implIntersect, implEqual, implEmpty, implIdealQuo.

Function: implUnion

Computes the union of algebraic sets.

Calling Sequence:

$U :=$ implUnion(As)

Parameters:

As : `exprseq(algset("impl"))`
Algebraic sets in implicit representation.

Result:

U : `algset("impl")`
The union of the given algebraic sets.

Description:

The function computes the union of algebraic sets in implicit form by computing the product of the corresponding ideals. These ideals are given by a finite basis. A basis of the product is obtained by forming all products of basis elements.
For the case of two algebraic sets A and B: Let the ideal of A have the basis $\{p_1, \ldots, p_n\}$ and let the ideal of B have the basis $\{q_1, \ldots, q_m\}$ then the product has the basis $\{p_1q_1, p_2q_1, \ldots, p_nq_1, \ldots, p_1q_m, p_2q_m, \ldots, p_nq_m\}$.
If the algebraic sets have common components then the result contains these components with higher multiplicity (compare implUnionLCM)

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^3 + x^2\, y - x,\ z],\ [x,\ y,\ z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x,\ y^2 + z^2 - 1],\ [x,\ y,\ z])$$

```
> implUnion(a1,a2);
```

$$\text{Implicit\_Algebraic\_Set}([x^4 + x^3\, y - x^2, x^3\, y^2 + z^2\, x^3 - x^3 + x^2\, y^3 + x^2\, y\, z^2 - x^2\, y - x\, y^2 - z^2\, x + x,\ x\, z,\ y^2\, z + z^3 - z],\ [x,\ y,\ z])$$

See Also:

implUnionLCM, implSubSet, implEqual, implEmpty, implIntersect, implIdealQuo, equalBaseSpaces.

Function: implUnionLCM

Compute the union of algebraic sets.

Calling Sequence:

$U :=$ implUnionLCM(As)

Parameters:

As : `exprseq(algset("impl"))`
algebraic sets in implicit representation

Result:

U : `algset("impl")`
The union of the given algebraic sets.

Description:

The function computes the union of algebraic sets in implicit form by computing the intersection of the corresponding ideals.
For the case of two algebraic sets A and B: Let the ideal of A be I, a subset of $K[x_1, \ldots, x_n]$, and let the ideal of B be J. The intersection is computed by the ideal contraction $S := \langle t * I \cup (t-1) * J \rangle \cap K[x_1, \ldots, x_n]$, where t is a new variable and $\langle X \rangle$ denotes the ideal generated by the set X. A basis for the ideal S can be obtained by a Gröbner basis calculation. Furthermore, $S = I \cap J$.
If the algebraic sets have common components then the multiplicity of these components is not increased in the result. (compare implUnion.)

Examples:

```
> a1 := mkImplAlgSet([x^3+x^2*y-x,z],[x,y,z]);
```

$$a1 := \mathrm{Implicit\_Algebraic\_Set}([x^3 + x^2\, y - x,\, z],\, [x,\, y,\, z])$$

```
> a2 := mkImplAlgSet([x,y^2+z^2-1],[x,y,z]);
```

$$a2 := \mathrm{Implicit\_Algebraic\_Set}([x,\, y^2 + z^2 - 1],\, [x,\, y,\, z])$$

```
> implUnionLCM(a1,a2);
```

$$\mathrm{Implicit\_Algebraic\_Set}([x^3 + x^2\, y - x,\, x\, z,\, y^2\, z + z^3 - z],$$
$$[x,\, y,\, z])$$

See Also:

implUnion, implSubSet, implEqual, implEmpty, implIntersect, implIdealQuo, equalBaseSpaces.

Function: tangSpace

Compute tangent space and tangent cone.

Calling Sequence:

$B :=$ tangSpace$(A, point)$
$B :=$ tangSpace$(A, point, \text{"}cone\text{"})$

Parameters:

A : `algset("impl")`

algebraic set in implicit representation

point : `list`

point in A.

"cone" : `string`

The string "cone".

Result:

B : `algset`

The tangent space of the given algebraic set in the given point.

Description:

The call tangSpace(A,*point*) computes an algebraic set defining the tangent space of variety A at point *point*. The tangent space is defined to be a set of points on all lines intersecting A at *point* with multiplicity bigger than 1. If the third optional parameter is the string "cone", then the tangent cone (set of lines generating the tangent space) is returned.

Examples:

```
> a1:=mkImplAlgSet([-15*y^2*z^3-76*y^3*z^2-z^5-3*y*z^4-15*x*y^2*z^2
> -11*x*y^3*z-x*z^4-3*x*y*z^3+y^3*x^2],[x,y,z],
> ["basespace"="projective"]);
```

$$a1 := \mathrm{Implicit\_Algebraic\_Set}([-15\,y^2\,z^3 - 76\,y^3\,z^2 - z^5 - 3\,y\,z^4 - 15\,x\,y^2\,z^2 - 11\,y^3\,z\,x - x\,z^4 - 3\,x\,y\,z^3 + y^3\,x^2],\,[x,\,y,\,z])$$

```
> tangSpace(a1,[0,1,0]);
```

$$\mathrm{Implicit\_Algebraic\_Set}([0],\,[x,\,y,\,z])$$

```
> tangSpace(a1,[0,1,0],"cone");
```

$$\mathrm{Implicit\_Algebraic\_Set}([-76\,z^2 - 11\,x\,z + x^2],\,[x,\,y,\,z])$$

See Also:

Groebnerbasis, homogeneousForm, leadingForm, numberOfTerms, pointInAlgSet, setRandomParameters.

Function: tsolve

Compute a triangularized representation.

Calling Sequence:

$B :=$ tsolve(A)

$B :=$ tsolve($A, vars$)

Parameters:

A : `algset("impl")`

An algebraic set in implicit representation.

vars : `list(name)`
A list of variables.

Result:

B : `exprseq(algset("impl"))`
union of several algebraic sets

Description:

The function splits an algebraic set in implicit form into the union of several algebraic sets which have a triangulized representation. The order of the variables in the variable list of the algebraic set influences the result. In the representation of every output algebraic set there are polynomials in just the last variable and bivariate polynomials in the last two variables.

Examples:

```
> a:=mkImplAlgSet([1-x+x*y^2+x*z^2,1-y+y*x^2+y*z^2,1-z+z*x^2+z*y^2],
> [x,y,z]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([1 - x + x\,y^2 + x\,z^2,\\ 1 - y + x^2\,y + y\,z^2,\ 1 - z + z\,x^2 + y^2\,z],\ [x,\ y,\ z])$$

```
> [tsolve(a,[x,y,z])];
```

$$[\mathrm{Implicit\_Algebraic\_Set}([\\ -1 - 2\,z + 2\,z^2 + 4\,z^3 - z^4 - 5\,z^5 + 2\,z^7, -3\,z^2 + 3\,y\,z^4\\ + 3\,y^2\,z^2 + 7\,z^4 - 2\,z^6\,y - 4\,z^4\,y^2 - 4\,z^6 + y^2\,z\\ - 6\,y\,z^2 - 2\,y\,z^3 + 2\,z^5\,y + y - z + z^3,\\ -1 + x - x\,y^2 - x\,z^2],\ [x,\ y,\ z]),\\ \mathrm{Implicit\_Algebraic\_Set}([\\ 4\,z^7 - 4\,z^6 - 4\,z^5 - 3\,z^3 + z^2 - z - 1, 16\,z^5\,y - 2\,z^5\\ + 16\,y\,z^4 - 6\,z^4 + 4\,y\,z^3 + 9\,z^3 + 2\,z^2 + 3\,z + 2 + 8\,z^6,\\ -1 + x - x\,y^2 - x\,z^2],\ [x,\ y,\ z])]$$

See Also:

casa.

Function: computeRadical

Computes a radical of an algebraic set.

Calling Sequence:

$R :=$ computeRadical(A)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation.

Result:

R : `exprseq(algset("impl"))`
The intersection of radical ideals.

Description:

The function splits the ideal of an algebraic set in implicit representation into the intersection of radical ideals. The ideal of an algebraic set is the set of all polynomials vanishing on the algebraic set.
First the ideal generated by the given basis and whose zeros form the given algebraic set is split at least into an intersection of unmixed-dimensional ideals. For each of the ideals a set of independent variables is computed along the splitting (that is why it is sometimes necessary to split unmixed-dimensional ideals, compare [GTZ88]). For each of the unmixed-dimensional ideals the radical is computed by reducing the problem to the zero-dimensional case.

Examples:

```
> a1:=mkImplAlgSet([z*x^2+z*y^2-z,x^4+2*x^2*y^2-2*x^2+y^4-2*y^2+1],
> [x,y,z]);
```

$$a1 := \mathrm{Implicit\_Algebraic\_Set}(\\ [z\,x^2 + y^2\,z - z,\ x^4 + 2\,y^2\,x^2 - 2\,x^2 + y^4 - 2\,y^2 + 1],\\ [x,\ y,\ z])$$

```
> [computeRadical(a1)];
```

$$[\mathrm{Implicit\_Algebraic\_Set}([x^2 + y^2 - 1,\ z],\ [x,\ y,\ z]),\\ \mathrm{Implicit\_Algebraic\_Set}([x^2 + y^2 - 1],\ [x,\ y,\ z])]$$

See Also:

mkImplAlgSet, toProj.

5.6 Operations on Algebraic Curves

Function: singularities

Compute singularities of a specified plane curve.

Calling Sequence:

$T :=$ singularities(A)

Parameters:

A : `planecurve`
Algebraic set in implicit representation specifying an irreducible plane curve.

Result:

T : `table`
A table of singularities of the given plane curve.

Description:

The function yields a table of singularities of the plane curve A.
Singularities are represented by a table T where T_i is a list of i-fold (projective) points on A.
This procedure does not compute neighboring singularities. Additional information on the trace of computation is provided by the userinfo routine.

Examples:

```
> a:=mkImplAlgSet([y^2*z^3-x^5],[x,y,z],["basespace"="projective"]);
```

$$a := \text{Implicit\_Algebraic\_Set}([y^2\, z^3 - x^5],\, [x,\, y,\, z])$$

```
> singularities(a);
```

table([
3 = [[0, 1, 0]]
2 = [[0, 0, 1]]
])

See Also:

neighbGraph.

Function: genus

Computes the genus of an algebraic set defining a plane curve.

Calling Sequence:

$g :=$ genus(A)

Parameters:

A : `planecurve`
A plane algebraic curve.

Result:

g : `integer`
The genus of the planecurve.

Description:

This procedure computes the genus g of an algebraic curve in implicit representation. An entry 'genus' = g is added to the list of attributes.
If the second argument *ntree* is used, it will be assigned the neighboring tree of the algebraic curve.

Examples:

```
> a:=mkImplAlgSet([y^3+y^2-x^2],[x,y]);
```

$$a := \text{Implicit\_Algebraic\_Set}([y^3 + y^2 - x^2],\, [x,\, y])$$

```
> genus(a);
```

0

See Also:

singularities, neighbGraph.

Function: imult

Intersection-multiplicity of two plane curves in affine space.

Calling Sequence:

$m := \mathsf{imult}(A, B, point)$

Parameters:

A : `planecurve`
algebraic set in implicit representation specifying a planecurve
B : `planecurve`
algebraic set in implicit representation specifying a planecurve
$point$: `list`
a point given as a list with two entries.

Result:

m : `integer`
the intersection multiplicity of he given curves in the given point

Description:

The funtion computes the intersection-multiplicity at *point* of the two plane curves.

Examples:

```
> a1:=mkImplAlgSet([x^4-y^2],[x,y]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([x^4 - y^2], [x, y])$$

```
> a2:=mkImplAlgSet([expand(x^4-y^3)],[x,y]);
```

$$a2 := \text{Implicit\_Algebraic\_Set}([x^4 - y^3], [x, y])$$

```
> imult(a1,a2,[0,0]);
```

$$8$$

See Also:

isProjective.

Function: adjointCurve

Computes an adjoint curve to a given curve.

Calling Sequence:

$C := \mathsf{adjointCurve}(A, deg, c)$

Parameters:

A : `algset`
An algebraic set defining a plane curve.

deg : `posint`
Degree of the adjoint curve

w : `name`
An unevaluated name to be used to construct names for parameters.

Result:

C : `polynom`
A linear system of plane curves.

Description:

The function computes a linear system C of plane curves passing through all singularities (including the neighboring ones) of a given plane curve A with multiplicities $m_P(c) \geq m_P(A) - 1$ for all c in C and all points P of the space. The argument holds upon return the coefficients of the adjoint curve.

Examples:

```
> a:=mkImplAlgSet([y^2*z^3-x^5],[x,y,z],["basespace"="projective"]);
```

$$a := \text{Implicit\_Algebraic\_Set}([y^2\, z^3 - x^5],\, [x,\, y,\, z])$$

```
> adjointCurve(a,3,w);
```

$$w_{0,\,1,\,2}\,(y-x)\,z^2 + w_{1,\,1,\,1}\,(x+y)\,(y-x)\,z + w_{2,\,0,\,1}\,(x+y)^2\,z + w_{3,\,0,\,0}\,(x+y)^3$$

See Also:

neighbGraph.

Function: neighbGraph

Computed the neighborhood graph of a plane algebraic curve.

Calling Sequence:

$l :=$ neighbGraph(A)

Parameters:

A : `algset("impl")`
An algebraic set in implicit representation specifying a plane curve.

Result:

l : `neighborhoodTree`
The neighborhood tree of the given plane curve.

Description:

This function returns the neighborhood tree of the given plane curve. It is represented by a list of form:
$l := []$ or $[node_1, ..., node_n]$ where

$$node_i := [mult, point, \{trans\}, neighborhoodTree]$$

The above structure has to be interpreted in the following way: On the top level, after applying the transformation trans in $node_i$, the top level points become the projective singularities of the curve. Each point is given its multiplicity. This linear coordinate transformation assures that $(1 : 0 : 0)$ and $(0 : 1 : 0)$ are not points on the homogenized function of the polynomial defining the plane curve. On the lower levels, we do the following: Move the singularity to the origin (i.e. $(0 : 0 : 1)$), apply the transformation trans in $node_i$ such that no tangent at $(0 : 0 : 1)$ is an irregular line (i.e. $x = 0$ or $y = 0$), no other fundamental point (i.e. $(1 : 0 : 0)$ and $(0 : 1 : 0)$) is on the curve. We get a transformed curve, say F. Now the points we have are the singularities (except $(0 : 1 : 0)$) of the quadratic transformation of F. Then the neighboring tree of all neighboring points is recursively computed.
For the theory behind neighboring points we refer to [Wal78].
Additional information on the trace of computation is provided by the userinfo routine.

Examples:

```
> a1:=mkImplAlgSet([y^2*z^3-x^5],[x,y,z],["basespace"="projective"]);
```

$$a1 := \text{Implicit\_Algebraic\_Set}([y^2\, z^3 - x^5],\ [x,\ y,\ z])$$

```
> neighbGraph(a1);
```

$$[[2,\ 0,\ 0,\ 1,\ \{x = x + y,\ y = y - x\},\ [[2,\ 1,\ 1,\ 0,\ \{\},\ []]]], [3,\ -1,\ 1,\ 0,\ \{x = x + y,\ y = y - x\},\ [[2,\ 2,\ 1,\ 0,\ \{x = x + 2\, y,\ y = y - 2\, x\},\ []]]]]$$

See Also:

adjointCurve.

Function: properParametrization

Returns a proper parametrization.

Calling Sequence:

$param :=$ properParametrization(A)

Parameters:

A : `algset("para")`
an algebraic set in parametric representation

Result:

$param$: `list(ratpoly)`

A proper parametrization for the given algebraic set.

Description:

The function computes a proper parametrization for an algebraic set in parametric representation. A proper parametrization fulfils the condition that the relationship between points on the curve and parameter values is one-to-one (with the exception of finitely many points on the curve).

Examples:

```
> a:=mkParaAlgSet([1/s^2,s^2],[s]);
```

$$a := \text{Parametric\_Algebraic\_Set}([\frac{1}{s^2},\, s^2],\, [s])$$

```
> properParametrization(a);
```

$$[\frac{-1+s}{-1+4\,s},\, \frac{-1+4\,s}{-1+s}]$$

See Also:

mkImplAlgSet, mkParaAlgSet, mkProjAlgSet, mkPlacAlgSet, reparametrize.

Function: passGenCurve

Generates a general curve and passing it through specified points.

Calling Sequence:

$C :=$ passGenCurve$(deg, points, c)$
$C :=$ passGenCurve$(deg, points, c, vars)$
$C :=$ passGenCurve$(deg, points, c, vars, ext)$

Parameters:

deg : `posint`
Degree of the resulting curve.
$points$: `list(list)`
List of points through which the curve should pass.
c : `table`
name of a global table to holding coefficients
$vars$: `list(name)`
List of variables.
ext : `list`
List of field extensions.

Result:

C : `algset`
A curve (or a system of curves) in projective space of degree deg passing through the specified points $points$.

Description:

The function returns a curve (or a system of curves) in projective 3-space of degree *deg* passing through specified *points*. The resulting curve will contain *vars* as variables.
The third parameter is used as a name of a table holding coefficients of the resulting curve (system).
If some points have algebraic coordinates, the fifth argument has to list extensions, where the coordinates of the points are lying. Precisely, for each point there has to be a list describing the extension of the coordinate field over the rational numbers. If all coordinates of a point are rational, the empty list has to be supplied. If some points have non-rational coordinates the procedure expects as the fifth argument a list of the same length as *points* where each element (list) specifies a tower of extensions $[\alpha_1, \ldots, \alpha_k]$ such that the corresponding point lies in $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ and the minimal polynomial for α_i (argument of the RootOf) is irreducible over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$.
A point in *points* is given either as a list $[x_0, x_1, x_2]$ (denoting a simple point) or as a nested list $[[x_0, x_1, x_2], m]$ (denoting a singular point with multiplicity m). Note that $m = 1$ is possible.

Examples:

```
> points := [[[RootOf(z^2+1),1,0],2],[[0,0,1],3],[[-1,-1,1],2]]:
> fieldExtensions := [[RootOf(z^2+1)],[],[]]:
> c1:=passGenCurve(5,points,u,[x,y,z],fieldExtensions);
```

$$\begin{aligned} c1 := \text{Implicit\_Algebraic\_Set}([&u_{[5,\,0,\,0]}\, x^5 + u_{[4,\,1,\,0]}\, x^4\, y \\ &+ 2\, u_{[5,\,0,\,0]}\, x^3\, y^2 + 2\, u_{[4,\,1,\,0]}\, x^2\, y^3 + u_{[5,\,0,\,0]}\, x\, y^4 \\ &+ u_{[4,\,1,\,0]}\, y^5 + u_{[4,\,0,\,1]}\, z\, x^4 \\ &+ (4\, u_{[5,\,0,\,0]} + 4\, u_{[4,\,1,\,0]} - u_{[2,\,2,\,1]})\, x^3\, y\, z \\ &+ u_{[2,\,2,\,1]}\, x^2\, y^2\, z \\ &+ (4\, u_{[5,\,0,\,0]} + 4\, u_{[4,\,1,\,0]} - u_{[2,\,2,\,1]})\, y^3\, z\, x \\ &+ (-u_{[4,\,0,\,1]} + u_{[2,\,2,\,1]})\, y^4\, z + (-4\, u_{[5,\,0,\,0]} \\ &- 2\, u_{[2,\,2,\,1]} + 4\, u_{[4,\,0,\,1]} + u_{[1,\,2,\,2]} + 2\, u_{[0,\,3,\,2]}) z^2\, x^3 \\ &+ (4\, u_{[4,\,1,\,0]} + 8\, u_{[5,\,0,\,0]} + 2\, u_{[2,\,2,\,1]} - 4\, u_{[4,\,0,\,1]} \\ &- 2\, u_{[1,\,2,\,2]} - 3\, u_{[0,\,3,\,2]}) x^2\, y\, z^2 + u_{[1,\,2,\,2]}\, x\, y^2\, z^2 \\ &+ u_{[0,\,3,\,2]}\, y^3\, z^2],\ [x,\ y,\ z]) \end{aligned}$$

```
> c2:=setRandomParameters(c1);
```

$$\begin{aligned} c2 := \text{Implicit\_Algebraic\_Set}([&4\, x^5 - 10\, x^4\, y + 8\, x^3\, y^2 \\ &- 20\, x^2\, y^3 + 4\, x\, y^4 - 10\, y^5 + 2\, z\, x^4 - 14\, x^3\, y\, z \\ &- 10\, x^2\, y^2\, z - 14\, y^3\, z\, x - 12\, y^4\, z + 8\, z^2\, x^3 \\ &- 27\, x^2\, y\, z^2 - 6\, x\, y^2\, z^2 + y^3\, z^2],\ [x,\ y,\ z]) \end{aligned}$$

```
> singularities(c2);
```

$$\begin{aligned} &\text{table}([\\ &3 = [[0,\ 0,\ 1]] \\ &2 = [[-1,\ -1,\ 1],\ [\text{RootOf}(\_Z^2 + 1),\ 1,\ 0]] \\ &]) \end{aligned}$$

See Also:

setRandomParameters, neighbGraph.

Function: rationalPoint

Compute a rational point on a conic.

Calling Sequence:

$r :=$ rationalPoint(f)

Parameters:

f : `polynom`
bivariate polynomial of total degree 2.

Result:

r : `list(string=anything)`
See description for details.

Description:

The function decides whether there is a rational point on the conic defined by f, and if so, computes one. The coefficients of f may be in Q or in Q(t) (and hence the coordinates of a rational point have to be in Q or in Q(t) respectively). Additional information is provided by rationalPoint: the decision whether f defines a parabola or not, the irreducibility of the conic, the time needed for the calculation, and g evaluated at the computed point (i.e. 0).

Examples:

```
> rationalPoint(x^2-4*x*y-3*y^2+4*x+8*y-5);
```

$$[\text{“Irreducible conic”} = \mathit{true},\ \text{“Parabola”} = \mathit{false},$$
$$\text{“Existence of a rational point”} = \mathit{true},$$
$$\text{“Its y-coordinate”} = 1,\ \text{“Its x-coordinate”} = 0,$$
$$\text{“Conic equation evaluated at this point”} = 0,$$
$$\text{“Time needed for calculation”} = 0]$$

See Also:

conic.

Function: conic

Tries to find a rational point on a conic.

Calling Sequence:

$F :=$ conic$(poly, vars, ok, parabol, ratpoint, X, Y)$;

Parameters:

poly : `polynom(rational)`
: Polynomial in two variables over $\mathbb{Q}$ respectively $\mathbb{Q}(t)$.

vars : `list`
: List of the two variables occuring in *poly*

ok : `boolean`
: Boolean variable indicating whether *poly* defines an irreducible conic

parabol : `boolean`
: Boolean variable indicating whether *poly* defines a parabola

ratpoint : `boolean`
: Boolean variable indicating whether there is a rational point on the conic defined by *poly*.

X : `name`
: First coordinate of the rational point on the conic if this exists.

Y : `name`
: Second coordinate of the rational point on the conic if this exists.

Result:

F : `function`
: the function conic returns itself, and stores all information in its parameters.

Description:

The call conic(*poly*,[x,y],*ok*,*parabol*,*ratpoint*,X,Y) decides whether there is a rational point on the conic defined by *poly*, and if so, computes one. The (boolean) value of the output parameter *ok* tells whether we deal with an irreducible conic. If $ok = false$ then the other output parameters should be ignored. Otherwise, the other four output parameters have the meaning declared above. If *poly* is defined over $\mathbb{Q}$ and no rational point lies on the conic, then (X, Y) represents the coordinates of a real point on the conic. In addition, if (X, Y) represents the coordinates of a point on the conic, then the procedure returns the result of substituting X, Y in *poly* $(= 0)$ as function value.

Examples:

```
> poly := x^2 - 4*x*y - 3*y^2 + 4*x + 8*y - 5:
> conic(poly,[x,y],'ok','parabol','ratpoint','X','Y');
```

$$0$$

```
> ok;
```

$$true$$

```
> [parabol, ratpoint, X, Y];
```

$$[false, true, 0, 1]$$

See Also:

rationalPoint.

5.7 Gröbner Bases

Function: GWalk

Computes the reduced Gröbner basis of the polynomial system by means of the Gröbner walk algorithm.

Calling Sequence:

$G :=$ GWalk$(F, X1)$
$G :=$ GWalk$(F, X1, X2)$

Parameters:

F : `list(polynom(C,` $X1$`))`
A list of polynomials in variables $X1$ over some coefficient domain C.

$X1$: `list(name)`
A list of variable names. The list of indeterminates $X1 = [x_1, \ldots, x_n]$ induces a lexicographical term order where $x_1 > x_2 > \ldots > x_n$ for which a Gröbner basis will be computed.

$X2$: `list(name)`
A list of the same variable names as occurring in $X1$ to specify the order of the variables for the intermediate computation of a total degree Groebner basis. The order of the indeterminates in $X2 = [x_1, x_2, \ldots, x_n]$ induces a total degree reverse lexicographical term order (tdeg) with $x_1 > x_2 > \ldots > x_n$. If the function is called with only two arguments, $X2$ defaults to $X1$.

Result:

G : `list(polynom(D,` $X1$`))`
A lexicographical Gröbner basis of F. The domain D is the quotient field of the domain C.

Description:

The function first computes a Gröbner basis with respect to the total degree order (tdeg) induced by the order of variables given in $X2$ by calling the function grobner[gbasis] and converts it to a lexicographical Groebner basis by means of the Groebner Walk algorithm as described in [CKM97].

Examples:

```
> GWalk([t^2+y^2+z^2-1, t^2+z^2-y, t-z], [t,y,z]);
```

$$[2\,z^2 - y,\, t - z,\, -4\,z^4 + 1 - 2\,z^2]$$

```
> GWalk([1+x^2+z,x*z+y^2*z],[z,x,y],[x,y,z]);
```

$$[1 + x^2 + z,\, y^2\,x^2 + x^3 + y^2 + x]$$

See Also:

Groebnerbasis, mgbasis, Groebner.

Function: mgbasis

Compute the normed reduced Gröbner basis.

Calling Sequence:

$GB :=$ mgbasis(F, X)
$GB :=$ mgbasis$(F, X, \mathit{torder1})$
$GB :=$ mgbasis$(F, X, \mathit{torder1}, \mathit{torder2})$

Parameters:

F : `list(list(polynom(rational)))`
A list of polynomial tuples.
X : `list(name)`
A list of indeterminates.
torder1 : `name`
A power-product-tuple ordering. Either term (for term first) or index (for index first - default).
torder2 : `name`
A power-product ordering. Either plex (for pure lexicographic) or tdeg (for total degree - default).

Result:

GB : `list(list(polynom(rational)))`
The normed reduced Gröbner basis of F.

Description:

The command mgbasis(F, X, *torder1*, *torder2*) computes the normed reduced Gröbner basis of F with respect to the indeterminates X and the given orderings. The algorithm used in mgbasis is described as Alg.13 in [Wal88].
The polynomial tuples in F are represented as lists of polynomials.
If X has the form $[x1, x2, \ldots, xn]$, then using the pure lexicographic ordering this is interpreted as $x1 > x2 > \ldots > xn$. Within the total degree ordering, ties are broken by inverse lexicographic order.

Examples:

```
> F := [[x*y-1,x+2], [y^2+x+1,y-1]]:
> mgbasis(F, [x,y]);
```

$$[[x^2 + x + y,\ -x - 2\,y],\ [x\,y - 1,\ x + 2],\ [y^2 + x + 1,\ y - 1],$$
$$[0,\ x^2 + x\,y + 2\,y^2 + 3\,x + y + 1]]$$

```
> mgbasis(F, [y,x], term, plex);
```

$$[[\frac{1}{4}\,x^3 + \frac{1}{4}\,x^2 + \frac{1}{4},\ y^2 + \frac{1}{2}\,y + \frac{1}{4}\,x^2 + \frac{5}{4}\,x],$$
$$[-\frac{1}{2}\,x^3 - \frac{1}{2}\,x^2 - \frac{1}{2},\ y\,x + \frac{1}{2}\,x^2 + \frac{1}{2}\,x + 1],$$
$$[y + x^2 + x,\ -2\,y - x]]$$

See Also:

mgbasis, mgbasisx, mnormalf, msolveGB, msolveSP.

Function: mgbasisx

Compute the normed reduced Gröbner basis along with the transformation matrix.

Calling Sequence:

$GB :=$ mgbasisx(F, G, GX, X)
$GB :=$ mgbasisx$(F, G, GX, X, \mathit{torder1})$
$GB :=$ mgbasisx$(F, G, GX, X, \mathit{torder1}, \mathit{torder2})$

Parameters:

F : `list(list(polynom(rational)))`
A list of polynomial tuples.

G : `list(list(polynom(rational)))`
The normed reduced Gröbner basis (output).

GX : `list(list(polynom(rational)))`
The transformation matrix (output), i.e. $GX * F = G$.

X : `list(name)`
A list of indeterminates.

torder1 : `name`
A power-product-tuple ordering. Either term (for term first) or index (for index first - default).

torder2 : `name`
A power-product ordering. Either plex (for pure lexicographic) or tdeg (for total degree - default).

Result:

GB : `list(list(polynom(rational)))`
The normed reduced Gröbner basis of F.

Description:

The command mgbasisx$(F, G, GX, X, \mathit{torder1}, \mathit{torder2})$ computes the normed reduced Gröbner basis of F with respect to the indeterminates X and the given orderings. The result is also returned in the variable G. In GX the transformation matrix is stored. The algorithm used in mgbasisx is basically the same as the one used in mgbasis, except that the reduction steps are stored to get the transformation matrix GX.
The polynomial tuples in F are represented as lists of polynomials.
If X has the form $[x1, x2, \ldots, xn]$, then using the pure lexicographic ordering this is interpreted as $x1 > x2 > \ldots > xn$. Within the total degree ordering, ties are broken by inverse lexicographic order.

Examples:

```
> F := [[x*y-1,x+2],[y^2+x+1,y-1]]:
> mgbasisx(F,'G','GX',[x,y]):
> G;
```

$$[[x^2+x+y,\ -x-2\,y],\ [x\,y-1,\ x+2],\ [y^2+x+1,\ y-1],\ [0,\ x^2+x\,y+2\,y^2+3\,x+y+1]]$$

```
> GX;
```

$$[[-y,\ x],\ [1,\ 0],\ [0,\ 1],\ [y^2+x+1,\ -x\,y+1]]$$

```
> mgbasisx(F,'G','GX',[y,x],term,plex):
> G;
```

$$[[\frac{1}{4}\,x^3+\frac{1}{4}\,x^2+\frac{1}{4},\ y^2+\frac{1}{2}\,y+\frac{1}{4}\,x^2+\frac{5}{4}\,x],\ [-\frac{1}{2}\,x^3-\frac{1}{2}\,x^2-\frac{1}{2},\ y\,x+\frac{1}{2}\,x^2+\frac{1}{2}\,x+1],\ [y+x^2+x,\ -2\,y-x]]$$

```
> GX;
```

$$[[\frac{1}{2}\,y^2+\frac{1}{4}+\frac{1}{2}\,x-\frac{1}{4}\,y\,x,\ -\frac{1}{2}\,y\,x+\frac{1}{2}+\frac{1}{4}\,x^2],\ [\frac{1}{2}+\frac{1}{2}\,y\,x,\ -\frac{1}{2}\,x^2],\ [-y,\ x]]$$

See Also:

mgbasis, mgbasisx, mnormalf, msolveGB, msolveSP.

Function: mnormalf

Compute the normal form of a tuple of polynomials modulo a module.

Calling Sequence:

$nf :=$ mnormalf(f, G, X)
$nf :=$ mnormalf($f, G, X, torder1$)
$nf :=$ mnormalf($f, G, X, torder1, torder2$)

Parameters:

f : `list(polynom(rational))`
A polynomial tuple.

G : `list(list(polynom(rational)))`
A list of polynomial tuples. Must be a Gröbner basis.

X : `list(name)`
A list of indeterminates.

torder1 : `name`
A power-product-tuple ordering. Either term (for term first) or index (for index first - default).

torder2 : `name`
A power-product ordering. Either plex (for pure lexicographic) or tdeg (for total degree - default).

Result:

nf : `list(polynom(rational))`
The normal form of f with respect to the Gröbner basis G.

Description:

The command mnormalf(f, G, X, *torder1*, *torder2*) computes the normal form of the tuple of polynomials f modulo the module generated by G with respect to the indeterminates X and the given orderings.
Note that G must be a Gröbner basis. Therefore, usually first a Gröbner basis for a set of polynomial tuples is computed (with mgbasis) and afterwards mnormalf is called.
If X has the form $[x1, x2, \ldots, xn]$, then using the pure lexicographic ordering this is interpreted as $x1 > x2 > \ldots > xn$. Within the total degree ordering, ties are broken by inverse lexicographic order.

Examples:

```
> F:=[[x-1,x*y-1],[x*y-1,y],[y,x-1]]:
> G1:=mgbasis(F,[x,y]):
> mnormalf([x^3*y^2+x^3+1,y^3+x+1],G1,[x,y]);
```

$$[0, x^2 + \frac{6}{5}\,x\,y + \frac{2}{5}\,y^2 - \frac{8}{5}\,x - \frac{2}{5}\,y - \frac{3}{5}]$$

```
> G2:=mgbasis(F,[x,y],term):
> mnormalf([x^3*y^2+x^3+1,y^3+x+1],G2,[x,y],term);
```

$$[-3\,x + \frac{3}{2}\,y + \frac{1}{2}, y^2 + \frac{3}{2}\,y]$$

See Also:

mgbasis, mgbasisx, mnormalf, msolveGB, msolveSP.

Function: msolveGB

Compute a basis for the module of syzygies.

Calling Sequence:

$S :=$ msolveGB(F, b, X)
$S :=$ msolveGB$(F, b, X, torder1)$
$S :=$ msolveGB$(F, b, X, torder1, torder2)$

Parameters:

F : `list(list(polynom(rational)))`
A list of polynomial tuples.
b : `list(polynom(rational))`
A polynomial tuple. (The right-hand-side of the linear system.)
X : `list(name)`
A list of indeterminates.

torder1 : `name`
A power-product-tuple ordering. Either term (for term first) or index (for index first - default).

torder2 : `name`
A power-product ordering. Either plex (for pure lexicographic) or tdeg (for total degree - default).

Result:

S : `[list(polynom(rational)), list(list(polynom(rational)))]`
A particular solution and a basis for the module of syzygies of the linear system $F * x = b$.

Description:

The command msolveGB(F, b, X, *torder1*, *torder2*) computes a basis for the module of syzygies using the extended Gröbner-basis-algorithm as described in [Wal88] (Alg.9 + Alg.13). The procedure returns a list with two elements: the first element is a particular solution, the second one is a basis for the module of syzygies (represented as a list).
The polynomial tuples in F are represented as lists of polynomials.
The variable F is interpreted as a matrix, e.g. F := [[f11,f21],[f12,f22],[f13,f23]] is a matrix with two rows and three columns.
If X has the form $[x1, x2, \ldots, xn]$, then using the pure lexicographic ordering this is interpreted as $x1 > x2 > \ldots > xn$. Within the total degree ordering, ties are broken by inverse lexicographic order.

Examples:

```
> F := [[x^3-y,-x+y],[x*y-1,x^3-1],[-x+y^2,y-1]]:
> S:=msolveGB(F,[0,0],[x,y]):
> nops(S[2]);
```

$$4$$

```
> mgbasis(S[2],[x,y]);
```

$$[[x^3\,y^2 - x^4 - x\,y^2 + x\,y - y^2 + x + y - 1,\\ x^3\,y - x^3 + x\,y^2 - y^3 - x^2 + x\,y - y^2 + y,\\ -x^6 + x^3\,y + x^3 - x^2\,y + x\,y^2 + x - 2\,y]]$$

```
> msolveGB(F,[x*y^2-y^2,-x^5+x^2-x+y^2],[x,y],term,tdeg);
```

$$[[y,\, -x^2,\, x], [[x^3\,y^2 - x^4 - x\,y^2 + x\,y - y^2 + x + y - 1,\\ x^3\,y - x^3 + x\,y^2 - y^3 - x^2 + x\,y - y^2 + y,\\ -x^6 + x^3\,y + x^3 - x^2\,y + x\,y^2 + x - 2\,y]]]$$

See Also:

mgbasis, mgbasisx, mnormalf, msolveGB, msolveSP.

Function: msolveSP

Compute a basis for the module of syzygies.

Calling Sequence:

$S :=$ msolveSP(F, b, X)
$S :=$ msolveSP$(F, b, X, \mathit{torder1})$
$S :=$ msolveSP$(F, b, X, \mathit{torder1}, \mathit{torder2})$

Parameters:

F : `list(list(polynom(rational)))`
A list of polynomial tuples.
b : `list(polynom(rational))`
A polynomial tuple. (The right-hand-side of the linear system.)
X : `list(name)`
A list of indeterminates.
torder1 : `name`
A power-product-tuple ordering. Either term (for term first) or index (for index first - default).
torder2 : `name`
A power-product ordering. Either plex (for pure lexicographic) or tdeg (for total degree - default).

Result:

S : `[list(polynom(rational)), list(list(polynom(rational)))]`
A particular solution and a basis for the module of syzygies of the linear system $F * x = b$.

Description:

The command msolveSP(F, b, X, *torder1*, *torder2*) computes a basis for the module of syzygies. This procedure is based on Thm.16 and Thm.11 in [Wal88]. The procedure returns a list with two elements: the first element is a particular solution, the second one is a basis for the module of syzygies (represented as a list).
The polynomial tuples in F are represented as lists of polynomials.
The variable F is interpreted as a matrix, e.g. $F :=$ [[f11,f21],[f12,f22],[f13,f23]] is a matrix with two rows and three columns.
If X has the form $[x1, x2, \ldots, xn]$, then using the pure lexicographic ordering this is interpreted as $x1 > x2 > \ldots > xn$. Within the total degree ordering, ties are broken by inverse lexicographic order.

Examples:

```
> F := [[x^3-y,-x+y],[x*y-1,x^3-1],[-x+y^2,y-1]]:
> S:=msolveGB(F,[0,0],[x,y]):
> nops(S[2]);
```

$$4$$

```
> mgbasis(S[2],[x,y]);
```

$$[[x^3\,y^2 - x^4 - x\,y^2 + x\,y - y^2 + x + y - 1,\\ x^3\,y - x^3 + x\,y^2 - y^3 - x^2 + x\,y - y^2 + y,\\ -x^6 + x^3\,y + x^3 - x^2\,y + x\,y^2 + x - 2\,y]]$$

```
> msolveGB(F,[x*y^2-y^2,-x^5+x^2-x+y^2],[x,y],term,tdeg);
```

$$[[y,\,-x^2,\,x],[[x^3\,y^2-x^4-x\,y^2+x\,y-y^2+x+y-1,$$
$$x^3\,y-x^3+x\,y^2-y^3-x^2+x\,y-y^2+y,$$
$$-x^6+x^3\,y+x^3-x^2\,y+x\,y^2+x-2\,y]]]$$

See Also:

mgbasis, mgbasisx, mnormalf, msolveGB, msolveSP.

5.8 Resultants

Function: mvresultant

Compute the resultant system of a system of multivariate polynomials.

Calling Sequence:

$r :=$ mvresultant($repr, vars$)

Parameters:

$repr$: `list(polynom(casaCoeffType, `$vars$`))`
list of polynomials in $vars$ with (optional) parameters and coefficients of type casaCoeffType

$vars$: `list(name)`
list of variables.

Result:

r : `list(polynom)`
the resultant system of the given system of multivariate polynomials

Description:

The function computes the resultant system of a system of multivariate polynomials in order to set up conditions for their solvability as well as formulas for calculating their solutions.
The vanishing of the resultant system provides a solvability criterion for the existance of common solutions of the system (in projective space).
We use the algorithm described in van der Waerden's Moderne Algebra, Springer-Verlag, 1994.

Examples:

```
> F1:=[x2^2+x1^2-1,x1*x2-1,x2^2 -x1]:
> mvresultant(F1,[x1,x2]);
```

$$[-2,\,1]$$

```
> F2:=[v*x2^2+RootOf(x^3+1)*x1^2-1,x1*x2-1,x2^2 -x1]:
> mvresultant(F2,[x1,x2]);
```

$$[-2\,\mathrm{RootOf}(\_Z^3+1)-2\,v+2, v^2+2\,v\,\mathrm{RootOf}(\_Z^3+1)$$
$$-2\,v+\mathrm{RootOf}(\_Z^3+1)^2-2\,\mathrm{RootOf}(\_Z^3+1)+1]$$

```
> F3:=[x3-x1*x2-x2,x2*x3-x1,x1*x3-x2]:
> mvresultant(F3,[x1]);
```

$$[x3-2\,x2,\ x2\,x3-1,\ x3-x2]$$

```
> mvresultant(F3,[x1,x2]);
```

$$[x3,\ 1-x3^2]$$

See Also:

resultant.

Function: subresultantChain

Compute the subresultant chain of two multivariate polynomials.

Calling Sequence:

$sc :=$ subresultantChain(P, Q, x)

Parameters:

P : `polynon`
A first polynomial

Q : `polynom`
A second polynomial

x : `name`
the variable over which the subresultants should be computed

Result:

sc : `list(polynom)`
A descending chain of all subresultants (including discrete ones) of the two polynomials P and Q.

Description:

This procedure computes the chain of subresultants of two given multivariate polynomials over a given variable. The list it returns contains all subresultants, both regular and discrete ones.

Examples:

```
> subresultantChain(x^5-2*x^3-5*x^2+x^3-2*x-5,x^2+1,x);
```

$$[x^5-x^3-5\,x^2-2\,x-5,\ x^2+1,\ 0,\ 0]$$

```
> subresultantChain(x^5-2*x^3-5*x^2+x^3-2*x-5,x^4+1,x);
```

$$[x^5-x^3-5\,x^2-2\,x-5,\ x^4+1,\ -5-x^3-5\,x^2-3\,x,$$
$$26+10\,x+22\,x^2,\ 180+120\,x,\ 1800]$$

See Also:

mvresultant.

5.9 Real Root Isolation

Function: realroot_a

Isolate real roots of a polynomial with algebraic number coefficients in strictly seperated intervals using de-recursive algorithms and norm.

Calling Sequence:

$r :=$ realroot_a$(minimal, interval, poly)$
$r :=$ realroot_a$(minimal, interval, poly, widthgoal)$

Parameters:

$minimal$: `polynom`
  the defining polynomial of an algebraic number α
$interval$: `list`
  an isolating interval for α
$poly$: `polynom`
  a square-free univariate polynomial with algebraic number coefficients (α).
$widthgoal$: `numeric`
  maximal size of each isolating interval.

Result:

R : `list`
  a sorted list of strictly separated isolating intervals for all real roots of the polynomial $poly$

Description:

The function returns a sorted list of strictly separated isolating intervals for all real roots of the polynomial $poly$.
The width of the interval is less than or equal to the optional parameter $widthgoal$ (a positive number). If $widthgoal$ is omitted, the most convenient width is used for each interval returned. The polynomial with algebraic number coefficients has to be square-free.

Examples:

```
> eq1 := expand(2*x^4 - 3*x^2*y + y^2*(y-1)^2);
```

$$eq1 := 2\,x^4 - 3\,x^2\,y + y^4 - 2\,y^3 + y^2$$

```
> Ay := 8*y^2 - 16*y - 1:
> a := RootOf(Ay):
> eq2 := subs(y=a, eq1):
> # Algebraic sqrfree
> Geq2 := evala(Gcd(eq2, diff(eq2, x))):
> evala(Divide(eq2, Geq2, evaln(eq2))):
> eq2 := primpart(subs(a=y, eq2));
```

$$eq2 := 4\,x^2 - 3\,y$$

```
> realroot_a(Ay, [-10,10], eq2);
```

$$[[\frac{-3}{2}, -1], [1, \frac{3}{2}]]$$

See Also:

realroot_sb, realroot.

Function: realroot_sb

Isolate real roots of a polynomial in strictly separated intervals using derecursive algorithms.

Calling Sequence:

$l :=$ realroot_sb($poly$)
$l :=$ realroot_sb($poly, widthgoal$)
$l :=$ realroot_sb($poly, interval$)
$l :=$ realroot_sb($poly, interval, widthgoal$)

Parameters:

poly : `polynom`
a univariate polynomial with numeric coefficients.
widthgoal : `number`
maximal size of each isolating interval,
interval : `list`
an interval where the user wants to isolate all of the real roots of the polynomial.

Result:

l : `list`
a sorted list.

Description:

The function returns a sorted list of strictly separated isolating intervals for all real roots of the univariate polynomial *poly* within *interval*. If *interval* is omitted, all realroots of the polynomial will be isolated (easier problem). The width of the interval is less than or equal to the optional parameter *widthgoal* - a positive number. If *widthgoal* is omitted, the most convenient width is used for each interval returned. The function now accepts polynomials with floating point coefficiences. The polynomial will be converted to a rational polynomial before isolating the realroots.
For isolating real roots within an OPEN interval, the algorithm in this function avoids to use transformations. It is many times faster than algorithms using transformations. For example with randpoly(x, $coeffs$= rand($-100 \ldots 100$), $terms$=10, $degree$=100), it is 20 times faster (see [Tra95b] for more details).

Examples:

```
> realroot_sb(2.5*x^8+x^7-4*x^6-20*x^5+4*x^4+20*x^3, 1/1000);
```

$$[[\frac{-493}{512}, \frac{-985}{1024}], [0, 0], [\frac{569}{512}, \frac{1139}{1024}], [\frac{951}{512}, \frac{1903}{1024}]]$$

```
> p1:=expand(subs({n=20,a=5},x^n-2*(a*x-1)^2)):
> realroot_sb(p1,[0,1]);
```

$$[[\frac{53687087}{268435456}, \frac{107374175}{536870912}], [\frac{13421773}{67108864}, \frac{6710887}{33554432}]]$$

See Also:

realroot_a, realroot.

5.10 Plotting

Function: plotAlgSet

Plots algebraic sets.

Calling Sequence:

$P :=$ plotAlgSet(A)
$P :=$ plotAlgSet$(A, options)$

Parameters:

A : `algset`
An algebraic set which is to be plotted.
options : `expseq`
Options analogous to the Maple plot options.

Result:

P : `PLOT`
A plotted graph.

Description:

The function plots the following algebraic sets: planar curves represented implicitly, in parametric form, or by places, space curves represented implicitly, in parametric form, in projected form, or by places, surfaces represented implicitly, or in parametric form.
Options are given as equations of the form $option = value$.
All options that are used by MAPLE's plotting-functions (plot[options]) are supported. Furthermore, the range for the variables can be given. For every variable occurring in the algebraic set the user must specify a range.
Also the option $option = n$ is supported. This option is used for plotting places. Each formal powerseries will be developed into a power series up to order n (default is the global variable $Order$).
Finally, the option $style = point$ is supported.

Examples:

```
> a1:=mkImplAlgSet([2*x^4-3*x^2*y+y^2*(y-1)^2],[x,y]);
```

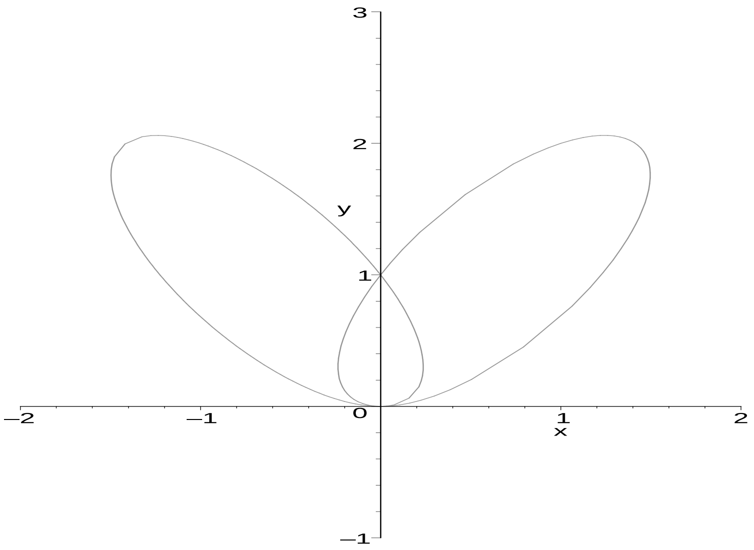
$$a1 := \text{Implicit\_Algebraic\_Set}([2\,x^4 - 3\,x^2\,y + y^2\,(y-1)^2],\ [x,\,y])$$

```
> plotAlgSet(a1,x=-2..2,y=-1..3,numpoints=200);
```

Time for isolating critical points : , .440

Time for finding intermediate points : , 5.701

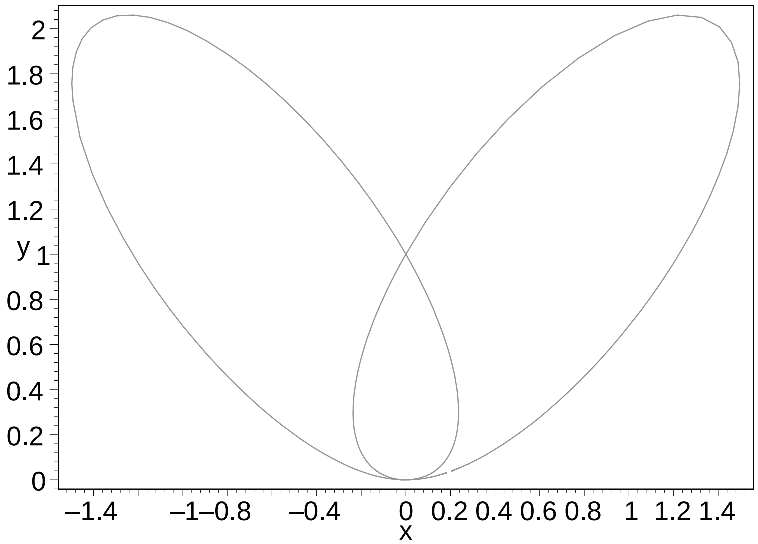
Time for others : , .030




```
> a2:=toPara(a1,t);
```

$$a2 := \text{Parametric\_Algebraic\_Set}([\frac{-16\,t - 53\,t^3 + 2 + 45\,t^2 + 22\,t^4}{-200\,t^3 + 3 + 114\,t^4 + 124\,t^2 - 32\,t},\ \frac{-6\,t + 1 + 4\,t^4 + 13\,t^2 - 12\,t^3}{-200\,t^3 + 3 + 114\,t^4 + 124\,t^2 - 32\,t}],\ [t])$$

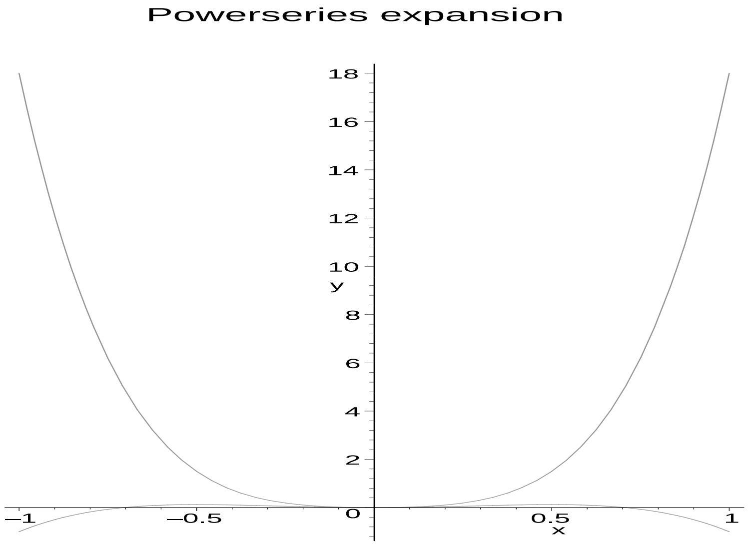
```
> plotAlgSet(a2,t=-10..10,scaling=constrained,axes=boxed);
```




```
> a3:=toPlac(a1,[0,0],T);
```

$$a3 := \mathrm{Places\_Algebraic\_Set}([[T, T^2 - 2\,T^4 + 17\,T^6 - 202\,T^8 + 2825\,T^{10} + \mathrm{O}(T^{11})], [T, 2\,T^2 + 16\,T^4 + 112\,T^6 + 1664\,T^8 + 15616\,T^{10} + \mathrm{O}(T^{11})]], [T])$$

```
> plotAlgSet(a3,T=-1..1,order=6,title='Powerseries expansion');
```



See Also:

ssiPlot, pacPlot.

Function: pacPlot

Plots plane algebraic curves using a hybrid symbolic-numerical method.

Calling Sequence:

$P :=$ pacPlot(A)
$P :=$ pacPlot$(A, options)$

Parameters:

A : `polynom`
A bivariate polynomial
options : `exprseq`
Options that are supported by MAPLE's plotting-functions plot[options].

Result:

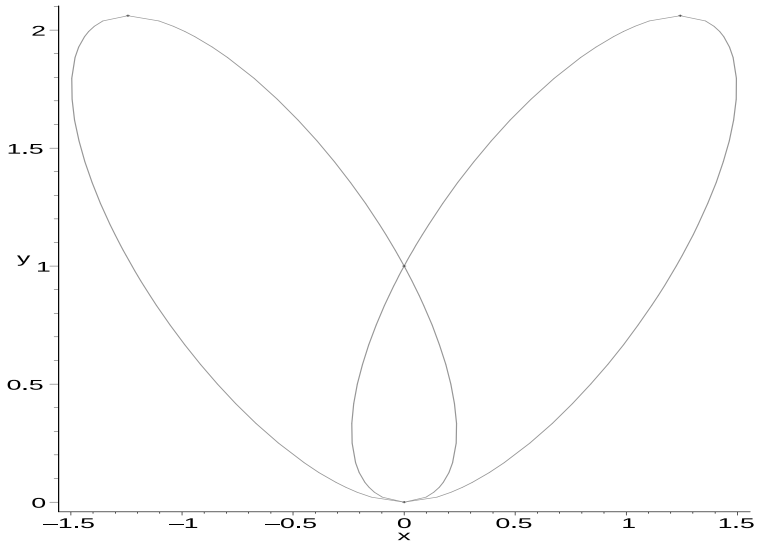
P : `PLOT`
A plot of the given algebraic curve

Description:

The function plots plane curves in implicit form. The range for the variables will be automatically choosen based on the critical points of the curve. Options are given as equations of the form *option* = *value*. All options that are supported by MAPLE's plotting-functions plot[options] are accepted.

Examples:

```
> pacPlot(2*x^4-3*x^2*y+y^2*(y-1)^2);
```



See Also:

ssiPlot, plotAlgSet, plot[options].

Function: ssiPlot

Plots the intersection of two surfaces using hybrid symbolic-numerical method.

Calling Sequence:

$P :=$ ssiPlot$(expr1, expr2, vars)$
$P :=$ ssiPlot$(expr1, expr2, vars, options)$

Parameters:

$expr1$: `polynom`
A polynomial in the three variables listed in $vars$.
$expr2$:
A polynomial in the three variables listed in $vars$.
$vars$: `list(name)`
list of variables
$options$: `exprseq`
options to the function

Result:

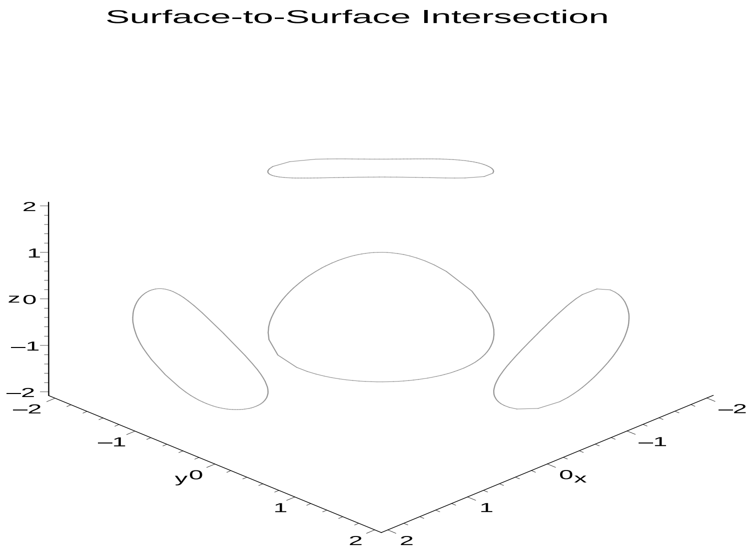
P : `PLOT3D`
a surface-to-surface intersection plot

Description:

The function plots the intersection of two three-dimensional and implicitly defined surfaces.
An appropriate range for each variable will be computed automatically. The user does not need to specify the ranges.
Options are given as equations of the form $option = value$.
All options that are available to MAPLE's plotting-functions (plot[$options$]) are supported.
Furthermore, the option $epsilon = -n$ Specifies precision for real root isolation procedures used in the ssiPlot function.
Finally, the option $maxiterate = n$ specifies the maximum number of extended Newton iterations used in the ssiPlot function.

Examples:

```
> ssiPlot(x^2+y^2+z^2 -5, x*y*z -1,[x,y,z],color=green);
```



See Also:

pacPlot, plotAlgSet.

5.11 Offset Curves

Function: implOffset

Computes the offset curve in implicit representation to a given curve.

Calling Sequence:

$S :=$ implOffset(C)
$S :=$ implOffset(C, d)
$S :=$ implOffset$(C, d, variables)$

Parameters:

C : {`algset("impl")`,`algset("para")`}
Algebraic curve in implicit or parametric representation,
d : {`name`,`rational` }
A variable or a rational number, specifying the distance of the offset curve to the original curve.
$variables$: `list(name)`
A list of two names for the representation of the resulting curve.

Result:

S : `algset("impl")`
The offset curve in implicit representation.

Description:

The function computes the offset curve in implicit representation at distance d to a given plane algebraic curve (in implicit or parametric representation). If the parameter d is not provided it is assumed that the distance is a parameter.
If the parameter *variables* is not provided and the curve is given in implicit form and the parameter d does not belong to the variable list of C then this variable list is taken as default.

Examples:

```
> a:=mkImplAlgSet([x^2-y],[x,y]);
```

$$a := \text{Implicit\_Algebraic\_Set}([x^2 - y],\, [x,\, y])$$

```
> implOffset(a,2);
```

$$\text{Implicit\_Algebraic\_Set}([\frac{225}{16}\,y^2 - 2\,x^2\,y^3 - \frac{5}{2}\,x^4\,y + x^6 \\ - 6\,y^2\,x^2 + \frac{15}{8}\,x^2\,y - \frac{191}{16}\,x^4 + y^4 + 43\,x^2 + x^4\,y^2 \\ - \frac{17}{2}\,y^3 + 34\,y - \frac{289}{4}],\, [x,\, y])$$

```
> a:=mkParaAlgSet([2*t/(t^2+1),(t^2-1)/(t^2+1)],[t]);
```

$$a := \text{Parametric\_Algebraic\_Set}([2\,\frac{t}{t^2+1},\, \frac{t^2-1}{t^2+1}],\, [t])$$

```
> [implOffset(a)];
```

$$[\text{Implicit\_Algebraic\_Set}([x^2 + 2\,z - z^2 - 1 + y^2],\, [x,\, y]), \\ \text{Implicit\_Algebraic\_Set}([x^2 - 2\,z - z^2 - 1 + y^2], \\ [x,\, y])]$$

```
> [implOffset(a,d)];
```

$$[\text{Implicit\_Algebraic\_Set}([x^2 - 1 - d^2 + 2\,d + y^2],\, [x,\, y]), \\ \text{Implicit\_Algebraic\_Set}([x^2 - 1 - 2\,d - d^2 + y^2], \\ [x,\, y])]$$

See Also:

mkImplAlgSet, mkParaAlgSet, paraOffset, RPHcurve.

Function: paraOffset

Compute the offset curve in parametric representation to a given curve.

Calling Sequence:

$S :=$ paraOffset(C)
$S :=$ paraOffset(C, d)
$S :=$ paraOffset$(C, d, variable)$

Parameters:

C : {`algset("impl")`,`algset("para")`}

Algebraic curve in implicit or parametric representation,

d : {`name,rational` }

A variable or a rational number, specifying the distance of the offset curve to the original curve.

variable : `name`

A variable name that is used for the resulting parametric representation.

Result:

S : `algset("para")`

The offset curve in parametric representation.

Description:

The function computes the offset curve in parametric representation at distance d to a given plane algebraic curve (in implicit or parametric representation).
If the parameter d is not provided it is assumed that the distance is a parameter.
If the parameter *variable* is not provided, the given algebraic curve is in parametric form and d is not its parameter then this parameter is taken.

Examples:

```
> a:=mkImplAlgSet([x^2-y],[x,y]);
```

$$a := \mathrm{Implicit\_Algebraic\_Set}([x^2 - y],\, [x,\, y])$$

```
> paraOffset(a,2);
```

$$\mathrm{Parametric\_Algebraic\_Set}([-\frac{1}{4}\,\frac{(x^2+1-8\,x)\,(-1+x^2)}{x\,(x^2+1)},\; \frac{1}{16}\,\frac{-x^2+1-x^4+x^6+64\,x^3}{x^2\,(x^2+1)}],\, [x])$$

```
> a:=mkParaAlgSet([2*t/(t^2+1),(t^2-1)/(t^2+1)],[t]);
```

$$a := \mathrm{Parametric\_Algebraic\_Set}([2\,\frac{t}{t^2+1},\, \frac{t^2-1}{t^2+1}],\, [t])$$

```
> [paraOffset(a)];
```

$$[\mathrm{Parametric\_Algebraic\_Set}([-2\,\frac{t\,(x-1)}{t^2+1},\, -\frac{(t^2-1)\,(x-1)}{t^2+1}],\, [t]),\, \mathrm{Parametric\_Algebraic\_Set}([2\,\frac{t\,(x+1)}{t^2+1},\, \frac{(t^2-1)\,(x+1)}{t^2+1}],\, [t])]$$

```
> [paraOffset(a,d)];
```

$$[\mathrm{Parametric\_Algebraic\_Set}([-2\,\frac{t\,(-1+d)}{t^2+1},\, -\frac{(t^2-1)\,(-1+d)}{t^2+1}],\, [t]),\, \mathrm{Parametric\_Algebraic\_Set}([2\,\frac{t\,(1+d)}{t^2+1},\, \frac{(t^2-1)\,(1+d)}{t^2+1}],\, [t])]$$

See Also:

mkImplAlgSet, mkParaAlgSet, implOffset, RPHcurve.

Function: RPHcurve

Checks if there exists an RPH (rational Phytagorean hodograph) parametrization of the given curve.

Calling Sequence:

$B :=$ RPHcurve(C, phi)

Parameters:

C : `algset("para")`
Rational curve in parametric representation.

phi : `name`
An unevaluated name which will store the reparametrizing function upon return.

Result:

B : `boolean`
The function returns true if the given curve has an RPH parametrization and false otherwise.

Description:

The function checks if there exists an RPH (Rational Phytagorean hodograph) parametrization of the algebraic curve C by analyzing the reducibility and rationality of the reparametrizing curve. If the reparametrizing curve is irreducible and rational then a quadratic reparametrizing function is assigned to the parameter phi.

Examples:

```
> a:=mkParaAlgSet([t^7,t^6],[t]);
```

$$a := \text{Parametric\_Algebraic\_Set}([t^7,\ t^6],\ [t])$$

```
> RPHcurve(a,p);
```

$$true$$

```
> p;
```

$$-\frac{12}{7}\,\frac{t}{t^2-1}$$

See Also:

mkImplAlgSet, mkParaAlgSet, paraOffset, implOffset.

5.12 Error Correcting Codes

Function: finiteCurve

The function prepares a curve in the projective plane over a finite field.

Calling Sequence:

$C :=$ finiteCurve$(f, Fq, indets)$

Parameters:

f : `polynom`
The polynom defining the curve. f will be homogenized if necessary.
Fq : `casa/finite/Field`
The ground field
$indets$: `list`
The indeterminants of f. This argument is optional.

Result:

C : `casa/finite/Curve`
The curve.

Description:

Constructs a curve in the projective plane over the field Fq. The defining polynomial is homogenized if necessary.
The elements of the curve can be accessed using.
op(1,C) is the defining polynomial,
op(2,C) is the list of indeterminants, and
op(3,C) is the field Fq.

Examples:

```
> C1 := finiteCurve( x^4 + y^3 + y, finiteField(3));
```

$$C1 := x^4 + y\,z^3 + y^3\,z$$

```
> C2 := finiteCurve( x^5 + y^5 + z^5, finiteField(4));
```

$$C2 := x^5 + y^5 + z^5$$

See Also:

finiteField.

Function: BCHDecode

Decode using error-locator decoding.

Calling Sequence:

$a :=$ BCHDecode(w, C)

Parameters:

w : `list, vector`
The received word.
C : `casa/finite/BCH2`
The 2d-BCH code.

Result:

a : `vector`
The message sent out.

Description:

Decodes a received word using Sakatas algorithm to find a basis for the error locator idea.
The vector returned is the message sent out, that is the solution of a * G = c, where G is the generator matrix of the code C, and c is the nearest neighbor of the word w.

Examples:

```
> F4:= finiteField(4);
```

$$F4 := Field$$

```
> C:= BCH2(3,3,[2,0],F4);
```

$$C := Cyclic/BCH2$$

```
> c:= CyclicEncode([seq(1,i=1..C["k"])],C);
```

$$c := [1, 1, 1, 1, 1, \alpha, 1, \alpha, 0]$$

```
> r:= evalm(c+[0,1,0,0,0,0,0,0,0]);
```

$$r := [1, 2, 1, 1, 1, \alpha, 1, \alpha, 0]$$

```
> BCHDecode(r,C);
```

$$[1, 1, 1, 1, 1, 1]$$

See Also:

finiteField, BCH2, CyclicEncode.

Function: BCH2

Prepare a 2-dimensional BCH-code.

Calling Sequence:

$C :=$ BCH2$(N1, N2, delta, F)$

Parameters:

$N1$: `integer`
The x value of the domain of C.
$N2$: `integer`
The y value of the domain of C.

delta : `list of integer`
  The parameter delta of the 2-dim BCH code.
F : `casa/finite/Field`
  The field over which C is defined.

Result:

C : `table`
  The BCH code of domain (N1,N2) with parameter delta over the field F

Description:

Prepares the 2-dimensional BCH-code C.
The code is a table with the following entries:
G: The generator matrix.
GIn: The generator matrix, in RootOf form.
H: The parity check matrix.
HIn: The parity check matrix, in RootOf form.
$infopos$: The set of info-positions.
$Roots$: The common roots of G.
n: The length of the code
k: The dimension of the code

Examples:

```
> C1 := BCH2(3,3,[1,0], finiteField(4));
```

$$C1 := Cyclic/BCH2$$

See Also:

finiteField.

Function: CyclicEncode

Encode using matrix multiplication.

Calling Sequence:

$c :=$ CyclicEncode(a, C)

Parameters:

a : `list, vector`
  The message to encode.
C : `casa/finite/Cyclic/BCH2`
  The code used for encoding

Result:

c : `vector`
  The coded message. c = a * G, where G is the generator matrix of the code C.

Description:

Encodes a message using the 2 dimensional cyclic code C. Encoding is done by a simple matrix multiplication.

Examples:

```
> F4:= finiteField(4);
```

$$F4 := Field$$

```
> C:=BCH2(3, 3, [4,0], F4);
```

$$C := Cyclic/BCH2$$

```
> CyclicEncode([seq(1, i=1..C["k"])], C);
```

$$[1, \alpha, \alpha^2, \alpha, \alpha^2, 1, \alpha^2, 1, \alpha]$$

See Also:

finiteField, BCH2.

Function: DivBasisL

Computes a basis of the space L(G).

Calling Sequence:

$B :=$ DivBasisL$(G, C, method)$
$B :=$ DivBasisL(G, C, f)

Parameters:

G : `casa/finite/divisor`
A basis of L(G) will be computed.
C : `casa/finite/curve`
The basis will be computed on the curve C.
The curve is assumed to be smooth. This is not checked by the function.
$method$: `string`
Algorithm used to determine the basis. Currently only "monomial" is availabe. This argument is optional.
f : {`list, set`}
Try to find a basis, as a subset of f.

Result:

B : `list(casa/finite/rational)`
A basis of the space L(G).

Description:

Compute a basis of the space L(G).
With method "monomial" only basis functions of the form $x^i y^j$ with $0 < i <$ deg(C, x) if $deg(C, x) > deg(C, y)$ otherwise the basis is of the same form, only with x and y exchanged.

If a list of basis functions is given, it is assumed, that the functions have poles at the Fq-rational points of the curve C only.
The Riemann-Roch theorem is used to test, whether the full basis is found or not.

Examples:

```
> C := finiteCurve(x^3+y^2+y, finiteField(4));
```

$$C := x^3 + y\,z^2 + y^2\,z$$

```
> H := makeDivisor( [ [0,1,0] ], [4] );
```

$$H := Divisor$$

```
> B := DivBasisL(H, C);
```

$$B := [1, \frac{y}{z}, \frac{x}{z}, \frac{x^2}{z^2}]$$

See Also:

finiteCurve, makeDivisor.

Function: finiteField

Constructs a finite field

Calling Sequence:

$F :=$ finiteField(q)
$F :=$ finiteField($q, alpha$)

Parameters:

q : `posint`
A prime power. The number of elements of the field.
$alpha$: `name`
Name used for a primite element of the field.

Result:

F : `casa/finite/Field`
A table with methods to work with finite fields.

Description:

finiteField constructs a table containing methods to convert elements of a finite field, represented using Maple's RootOf expressions, to a more readable form α^i. Also conversion to RoofOf form, and routines to normalize, expand and normalize expressions containing RootOfs.
The following methods are supplied:
p, k, q: $q = p^k$. The size of the field.
In: Convert an expression using alphas to RootOf form.
Out: Convert from RootOf form to form using alphas. If the conversion is not possible, then the argument is returned unchanged.

N: Normalize a field element using the inert Normal function.
mapN: Map the normalization function N to a list, set, ...
Expand: Expand a field element using the inert Expand function.
seq: A sequence of the field elements, in alpha notation.
seqIn: A sequence of the field elements, represented using RootOfs.
mapIn, *mapOut*: Map the conversion functions *In* and *Out* to a list, set, etc.
mapOutIn: This function is equivalent to *mapOut* followed by *mapIn*. This is useful to do operations on vectors stored in alpha form.

Examples:

```
> F := finiteField(16):
> F["p"];
```

$$2$$

```
> F["q"];
```

$$16$$

```
> F["k"];
```

$$4$$

```
> F["seq"];
```

$$[0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}]$$

```
> a := F["In"](alpha^2 + 1);
```

$$a := \mathrm{RootOf}(\_Z^4 + \_Z + 1)^2 + 1$$

```
> F["Out"](1 + a + a^2);
```

$$\alpha^5$$

```
> a := F["In"](alpha^4 + alpha + 1);
```

$$a := \mathrm{RootOf}(\_Z^4 + \_Z + 1)^4 + \mathrm{RootOf}(\_Z^4 + \_Z + 1) + 1$$

```
> F["N"](a);
```

$$0$$

```
> F["Out"](a);
```

$$0$$

See Also:

GF, RootOf, Expand, Normal, evala, map.

Function: GoppaDecode

Decode by using error-locator decoding.

Calling Sequence:

$a :=$ GoppaDecode(w, C, EL)

Parameters:

w : {`list,vector`}
The received word
C : `casa/finite/Goppa/primary`

The primary Goppa code

EL : `function`

The error locator used for decoding.

Result:

a : `vector`

The message sent out.

Description:

Decodes a received word using error locator decoding, that is either the SV or Duursma-Algorirhm. The error locator must be initialized with GoppaPrepareSV or GoppaPrepareDu and then specified as the third argument.
The vector returned is the message sent out, that is the soluton of a * G = c, where G is the generator matrix of the code C, and c is the nearest neighbor of the word w.

Examples:

```
> C1 := finiteCurve(x^3 + y^2 + y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4):
```

$$G = +4\ (0 : 1 : 0)$$

$$\begin{aligned} D\ &= [0,\,0,\,1]\ \ +[0,\,1,\,1]\ \ +[1,\,\alpha,\,1]\ \ +[1,\,\alpha^2,\,1] \\ &+[\alpha,\,\alpha,\,1]\ \ +[\alpha,\,\alpha^2,\,1]\ \ +[\alpha^2,\,\alpha,\,1] \\ &+[\alpha^2,\,\alpha^2,\,1] \end{aligned}$$

$$L(G) = \mathrm{span}(1,\,\frac{y}{z},\,\frac{x}{z},\,\frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n,\,k,\,d_\Gamma] = 8,\,4,\,4$$

```
> w := [ 1 $ H1["n"] ];
```

$$w := [1,\,1,\,1,\,1,\,1,\,1,\,1,\,1]$$

```
> Du := GoppaPrepareDu(H1):
```

$$G\ +\ gQ = +5\ (0 : 1 : 0)$$

$$L(G+gQ) = \mathrm{span}(1,\,\frac{x}{z},\,\frac{y}{z},\,\frac{x^2}{z^2},\,\frac{x\,y}{z^2})$$

$$\psi * \chi = \begin{bmatrix} 1 & \frac{x}{z} \\ \frac{x}{z} & \frac{x^2}{z^2} \end{bmatrix}$$

$$\text{order}(\psi * \chi) = \left[\begin{array}{cc} 0 & 2 \\ 2 & 4 \end{array}\right]$$

```
> GoppaDecode(w, H1, Du);
```

$$S = \left[\begin{array}{cc} 0 & 0 \\ 0 & ?_{2,\,2} \end{array}\right]$$

$$e * \phi = [0, 0, 0, ?_4, ?_5]$$

$$S = \left[\begin{array}{cc} 0 & 0 \\ 0 & ?_{2,\,2} \end{array}\right]$$

$$e * \phi = [0, 0, 0, ?_4, ?_5]$$

$$\mathit{Nullspace}(S) = ([1])$$

$$\mathit{theta} = 1$$

$$\mathit{error\ locations} = \{\}$$

$$\mathit{Error\ Value\ System} : \text{matrix}([[], [], []]) * e = \left[\begin{array}{c} 0 \\ 0 \\ 0 \end{array}\right]$$

$$e = []$$

$$c = [1, 1, 1, 1, 1, 1, 1, 1]$$

$$[1, 1, 1, 1]$$

See Also:

GoppaPrimary, GoppaEncode, GoppaPrepareSV, GoppaPrepareDu.

Function: GoppaEncode

Encodes by using matrix multiplication.

Calling Sequence:

$c :=$ GoppaEncode(a, C)

Parameters:

a : {`list`,`vector`}
The message to encode.
C : {`casa/finite/Goppa/primary`,`casa/finite/Goppa/functional`}
The code used for encoding.

Result:

c : `vector`
The coded message. c = a * G, where G is the generator matrix of the code C.

Description:

Encodes a message using the Goppa codes C. Encoding is done by a simple matrix multiplication.

Examples:

```
> C1 := finiteCurve(x^3+y^2+y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4):
```

$$G = +4\ (0 : 1 : 0)$$

$$\begin{aligned} D\ &= [0,\,0,\,1]\ \ +[0,\,1,\,1]\ \ +[1,\,\alpha,\,1]\ \ +[1,\,\alpha^2,\,1] \\ &+[\alpha,\,\alpha,\,1]\ \ +[\alpha,\,\alpha^2,\,1]\ \ +[\alpha^2,\,\alpha,\,1] \\ &+[\alpha^2,\,\alpha^2,\,1] \end{aligned}$$

$$L(G) = \mathrm{span}(1,\, \frac{y}{z},\, \frac{x}{z},\, \frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n,\,k,\,d_\Gamma] = 8,\,4,\,4$$

```
> a := [ 1 $ H1["k"] ];
```

$$a := [1,\,1,\,1,\,1]$$

```
> c := GoppaEncode(a, H1);
```

$$c := [1,\,1,\,1,\,1,\,1,\,1,\,1,\,1]$$

See Also:

GoppaPrimary, GoppaDecode.

Function: GoppaPrepareDu

Initializes a Dursmaa error locator.

Calling Sequence:

Du := GoppaPrepareDu($Code$)
Du := GoppaPrepareDu($Code, baseMethod$)

Parameters:

$Code$: `casa/finite/Goppa/primary`
A one point primary Goppa code
$baseMethod$: `string`
Method for determining basis needed, passed to DivBasisL.

Result:

Du : `function`

A function, finding the error locations, the test functions and unknown syndromes for decoding.

Description:

Performed the initializaton step for the Duursma algorithm, used for finding the error locations of a primary Goppa code.

Examples:

```
> C1 := finiteCurve(x^3 + y^2 + y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4):
```

$$G = +4\ (0 : 1 : 0)$$

$$\begin{aligned} D\ &= [0,\,0,\,1]\ \ +[0,\,1,\,1]\ \ +[1,\,\alpha,\,1]\ \ +[1,\,\alpha^2,\,1] \\ &+[\alpha,\,\alpha,\,1]\ \ +[\alpha,\,\alpha^2,\,1]\ \ +[\alpha^2,\,\alpha,\,1] \\ &+[\alpha^2,\,\alpha^2,\,1] \end{aligned}$$

$$L(G) = \operatorname{span}(1,\,\frac{y}{z},\,\frac{x}{z},\,\frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n,\,k,\,d_\Gamma] = 8,\,4,\,4$$

```
> SV := GoppaPrepareDu(H1):
```

$$G\ +\ gQ = +5\ (0 : 1 : 0)$$

$$L(G + gQ) = \operatorname{span}(1,\,\frac{x}{z},\,\frac{y}{z},\,\frac{x^2}{z^2},\,\frac{x\,y}{z^2})$$

$$\psi\,*\,\chi = \begin{bmatrix} 1 & \frac{x}{z} \\ \frac{x}{z} & \frac{x^2}{z^2} \end{bmatrix}$$

$$\operatorname{order}(\psi\,*\,\chi) = \begin{bmatrix} 0 & 2 \\ 2 & 4 \end{bmatrix}$$

```
> w := [1 $ H1["n"] ];
```

$$w := [1,\,1,\,1,\,1,\,1,\,1,\,1,\,1]$$

```
> c := GoppaDecode(w, H1, SV);
```

$$S = \begin{bmatrix} 0 & 0 \\ 0 & ?_{2,\,2} \end{bmatrix}$$

$$e\,*\,\phi = [0,\,0,\,0,\,?_4,\,?_5]$$

$$S = \begin{bmatrix} 0 & 0 \\ 0 & ?_{2,\,2} \end{bmatrix}$$

$$e\,*\,\phi = [0,\,0,\,0,\,?_4,\,?_5]$$

$$Nullspace(S) = ([1])$$
$$theta = 1$$
$$error\ locations = \{\}$$
$$Error\ \ Value\ System: \text{matrix}([[], [], []]) * e = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$
$$e = []$$
$$c = [1, 1, 1, 1, 1, 1, 1, 1]$$
$$c := [1, 1, 1, 1]$$

See Also:

GoppaPrimary, GoppaDecode, DivBasisL, GoppaPrepareSV.

Function: GoppaPrepareSa

Initialize Sakata's procedure for 1-point AG-codes.

Calling Sequence:

Sak := GoppaPrepareSa($Code$)
Sak := GoppaPrepareSa($Code, baseMethod$)

Parameters:

$Code$: `casa/finite/Goppa/primary`
A one point primary Goppa code whose nongap semi- group is spanned by 2 elements.
$baseMethod$: `string`
Method for determining basis needed, passed to DivBasisL.

Result:

Sak : `function`
A list containing all information you need for decoding using Sakata's algorithm which can be computed once for a code. .

Description:

Performes the initializaton step for using Sakata's algorithm for one-point codes.
The current implementation is only for one point Goppa codes with 2 generaters. The returned function can be used with SakataDecode.

Examples:

```
> infolevel[‘casa/finite‘] := 5;
```

$$infolevel_{casa/finite} := 5$$

```
> C1 := finiteCurve(x^3 + y^2 + y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4);
```

$$G = +4\ (0 : 1 : 0)$$

$$D = [0, 0, 1] + [0, 1, 1] + [1, \alpha, 1] + [1, \alpha^2, 1] + [\alpha, \alpha, 1] + [\alpha, \alpha^2, 1] + [\alpha^2, \alpha, 1] + [\alpha^2, \alpha^2, 1]$$

$$L(G) = \operatorname{span}(1, \frac{y}{z}, \frac{x}{z}, \frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n, k, d_\Gamma] = 8, 4, 4$$

$$H1 := Goppa/primary$$

```
> Sak := GoppaPrepareSa(H1):
```

$$\text{“a”} = 2$$

$$\text{“The corresponding function”} = \frac{x}{z}$$

$$\text{“b”} = 3$$

$$\text{“The corresponding function”} = \frac{y}{z}$$

$$\frac{x^3}{z^3} = \frac{y}{z} + \frac{y^2}{z^2}$$

$$\text{“result”} = \begin{bmatrix} 1 & \frac{x}{z} & \frac{x^2}{z^2} & \frac{x^3}{z^3} \\ \frac{y}{z} & \frac{x\,y}{z^2} & \frac{x^2\,y}{z^3} & \frac{x^3\,y}{z^4} \\ \frac{y^2}{z^2} & \frac{x\,y^2}{z^3} & \frac{x^2\,y^2}{z^4} & \frac{x^3\,y^2}{z^5} \end{bmatrix}$$

$$\text{order}(\text{“result”}) = \begin{bmatrix} 0 & 2 & 4 & 6 \\ 3 & 5 & 7 & 9 \\ 6 & 8 & 10 & 12 \end{bmatrix}$$

```
> w := [1 $ (H1["n"]-1), 0 ];
```

$$w := [1, 1, 1, 1, 1, 1, 1, 0]$$

```
> c := SakataDecode(w, H1, Sak);
```

$$\text{“S”} = \begin{bmatrix} 1 & \alpha^2 & \alpha \\ \alpha^2 & S_{2,2} & \\ & & \end{bmatrix}$$

“Sakata’s algorithm for known syndroms:”

$$\text{“Minimal set generating the syndromarray up to”} = 4$$

$$\text{“F”} = [\alpha^2 + y, \alpha^2 + x]$$

$$\text{“Starting majority voting for sydroms of order”} = 5$$

$$\text{“The points of this order are”} = [[1,\ 1]]$$

$$\text{“[votes,multiplicities]”} = ([\alpha,\ 2])$$

$$\text{“S”}_{2,\,2} = \alpha$$

“Running Sakata's algorithm for the new known sy \
ndroms”

“Minimal set generating the syndromarray up to the \
order” = 5

$$\text{“F”} = [\alpha^2 + y,\ \alpha^2 + x]$$

$$\text{“S”} = \begin{bmatrix} 1 & \alpha^2 & \alpha \\ \alpha^2 & \alpha & \\ & & \end{bmatrix}$$

$$\text{“poss. error positions”} = [1,\ 2,\ 7,\ 8]$$

$$\mathit{Error\ \ Value\ \ System} : \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & \alpha^2 & \alpha^2 \\ 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & \alpha & \alpha \\ 0 & 0 & 1 & \alpha \end{bmatrix} * e = \begin{bmatrix} 1 \\ \alpha^2 \\ \alpha^2 \\ \alpha \\ \alpha \end{bmatrix}$$

$$e = [0,\ 0,\ 0,\ 1]$$

$$c = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]$$

$$c := [1,\ 1,\ 1,\ 1]$$

See Also:

GoppaPrimary, SakataDecode, DivBasisL, GoppaPrepareDu, GoppaPrepareSV.

Function: GoppaPrepareSV

Initializes a Skorobotatov-Vladut error locator.

Calling Sequence:

SV := GoppaPrepareSV(C)

Parameters:

C : `casa/finite/Goppa/primary`
A primary Goppa code.

Result:

SV : `function`
An error locator, using the Skorobotatov-Vladut algorithm.

Description:

Prepares an error locator for the Skorobotatov-Vladut algorithm, for the given primary Goppa code.

Examples:

```
> C1 := finiteCurve(x^3+y^2+y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4):
```

$$G = +4\ (0 : 1 : 0)$$

$$\begin{aligned} D\ &= [0, 0, 1]\ + [0, 1, 1]\ + [1, \alpha, 1]\ + [1, \alpha^2, 1] \\ &+ [\alpha, \alpha, 1]\ + [\alpha, \alpha^2, 1]\ + [\alpha^2, \alpha, 1] \\ &+ [\alpha^2, \alpha^2, 1] \end{aligned}$$

$$L(G) = \mathrm{span}(1, \frac{y}{z}, \frac{x}{z}, \frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n, k, d_\Gamma] = 8, 4, 4$$

```
> SV := GoppaPrepareSV(H1):
```

$$A = +2\ (0 : 1 : 0)$$

$$C = +2\ (0 : 1 : 0)$$

$$\psi = [1, \frac{x}{z}]$$

$$\chi = [1, \frac{x}{z}]$$

$$\psi * \chi = \begin{bmatrix} 1 & \frac{x}{z} \\ \frac{x}{z} & \frac{x^2}{z^2} \end{bmatrix}$$

```
> w := [1 $ H1["n"] ];
```

$$w := [1, 1, 1, 1, 1, 1, 1, 1]$$

```
> c := GoppaDecode(w, H1, SV);
```

$$S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\mathit{Nullspace}(S) = ([0, 1], [1, 0])$$

$$\mathit{error\ locations} = \{1, 2\}$$

$$\mathit{Error\ Value\ System} : \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} * e = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$e = [0, 0]$$

$$c = [1, 1, 1, 1, 1, 1, 1, 1]$$

$$c := [1, 1, 1, 1]$$

See Also:

GoppaPrimary, GoppaDecode, GoppaPrepareDu.

Function: GoppaPrimary

Prepares a primary Goppa code.

Calling Sequence:

$C :=$ GoppaPrimary(C, D, G)
$C :=$ GoppaPrimary$(C, D, G, basis)$

Parameters:

C : `casa/finite/Curve`
The curve to define the code.

D : {`casa/finite/Divisor, list, string`}
Points used to define the code. This can be a divisor, or a list of the points, or a string.
The following strings are accepted:
"affine": The affine Fq-rational points of the curve.
"rest": The Fq-rational points of the curve, except those used by the divisor G.

G : {`casa/finite/Divisor, integer`}
Divisor used to define the code. If only an integer is specified, and the curve has only a single pont at infinty, then this is value taken to be the multiplicity of the single point at infinity.

$basis$: {`string, list`}
Method to determine the basis of L(G). Passed as additional argument to DivBasisL.

Result:

C : `casa/finite/Goppa/primary`
The code CO(D,G).

Description:

Prepares the primary Goppa code CO(D,G).
The code is a table with the following entries:
G: The generator matrix.
GIn: The generator matrix, in RootOf form.
H: The pariy check matrix.
HIn: The pary check matrix, in RootOf form.
C: The curve defining the code.
LG: The divisor G and a basis of the space. The divisor can be accessed as C[LG][1], the basis using C[LG][2].
P: The support of the divisor D, that is the points on which the rational function in L(G) are evaluated.
n: The length of the code

k: The dimension of the code
$d[Gamma]$, d_Gamma: The designed distance of the code

Examples:

```
> C1 := finiteCurve(x^3+y^2+y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> P1 := GoppaPrimary(C1, "affine", 4);
```

$$G = +4\ (0 : 1 : 0)$$

$$\begin{aligned} D\ &= [0,\,0,\,1]\ \ + [0,\,1,\,1]\ \ + [1,\,\alpha,\,1]\ \ + [1,\,\alpha^2,\,1] \\ &+ [\alpha,\,\alpha,\,1]\ \ + [\alpha,\,\alpha^2,\,1]\ \ + [\alpha^2,\,\alpha,\,1] \\ &+ [\alpha^2,\,\alpha^2,\,1] \end{aligned}$$

$$L(G) = \mathrm{span}(1,\,\frac{y}{z},\,\frac{x}{z},\,\frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n,\,k,\,d_\Gamma] = 8,\,4,\,4$$

$$P1 := Goppa/primary$$

See Also:

GoppaEncode, GoppaDecode, DivBasisL.

Function: SakataDecode

Decode using Sakata's algorithm.

Calling Sequence:

$a :=$ SakataDecode(w, C, EL, way)
$a :=$ SakataDecode(w, C, EL)

Parameters:

w : `list, vector`
The received word
C : `casa/finite/Goppa/primary`
The primary Goppa code (one point with 2 nongaps)
EL : `function`
The initialisation of the Sakata algorithm: GoppaPrepareSa(C)
way : `string`
Determines whether decoding should be done by finding roots ("root" - the default of an error-locator or determining all syndroms ("syndrom").

Result:

a : `vector`
The message sent out.

Description:

Decodes a received word using Sakata's approach. Where all steps which must be computed once are computed by GoppaPrepareSa
The vector returned is the most probable message sent out.

Examples:

```
> infolevel[`casa/finite`] := 5;
```

$$infolevel_{casa/finite} := 5$$

```
> C1 := finiteCurve(x^3 + y^2 + y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 4);
```

$$G = +4\ \ (0 : 1 : 0)$$

$$D\ = [0, 0, 1]\ + [0, 1, 1]\ + [1, \alpha, 1]\ + [1, \alpha^2, 1]\ + [\alpha, \alpha, 1]\ + [\alpha, \alpha^2, 1]\ + [\alpha^2, \alpha, 1]\ + [\alpha^2, \alpha^2, 1]$$

$$L(G) = \mathrm{span}(1, \frac{y}{z}, \frac{x}{z}, \frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$[n, k, d_\Gamma] = 8, 4, 4$$

$$H1 := Goppa/primary$$

```
> w := [ 1 $ (H1["n"]-1), 0 ];
```

$$w := [1, 1, 1, 1, 1, 1, 1, 0]$$

```
> Sa := GoppaPrepareSa(H1):
```

$$\text{“a”} = 2$$

$$\text{“The corresponding function”} = \frac{x}{z}$$

$$\text{“b”} = 3$$

$$\text{“The corresponding function”} = \frac{y}{z}$$

$$\frac{x^3}{z^3} = \frac{y}{z} + \frac{y^2}{z^2}$$

$$\text{“result”} = \begin{bmatrix} 1 & \frac{x}{z} & \frac{x^2}{z^2} & \frac{x^3}{z^3} \\ \frac{y}{z} & \frac{x\,y}{z^2} & \frac{x^2\,y}{z^3} & \frac{x^3\,y}{z^4} \\ \frac{y^2}{z^2} & \frac{x\,y^2}{z^3} & \frac{x^2\,y^2}{z^4} & \frac{x^3\,y^2}{z^5} \end{bmatrix}$$

$$\text{order(“result”)} = \begin{bmatrix} 0 & 2 & 4 & 6 \\ 3 & 5 & 7 & 9 \\ 6 & 8 & 10 & 12 \end{bmatrix}$$

```
> SakataDecode(w, H1, Sa, "root");
```

$$\text{“S”} = \begin{bmatrix} 1 & \alpha^2 & \alpha \\ \alpha^2 & S_{2,\,2} & \\ & & \end{bmatrix}$$

“Sakata's algorithm for known syndroms:”

“Minimal set generating the syndromarray up to” $= 4$

$$\text{“F”} = [\alpha^2 + y,\ \alpha^2 + x]$$

“Starting majority voting for sydroms of order” $= 5$

“The points of this order are” $= [[1,\ 1]]$

“[votes,multiplicities]” $= ([\alpha,\ 2])$

$$\text{“S”}_{2,\,2} = \alpha$$

“Running Sakata's algorithm for the new known sy \
ndroms”

“Minimal set generating the syndromarray up to the \
order” $= 5$

$$\text{“F”} = [\alpha^2 + y,\ \alpha^2 + x]$$

$$\text{“S”} = \begin{bmatrix} 1 & \alpha^2 & \alpha \\ \alpha^2 & \alpha & \\ & & \end{bmatrix}$$

“poss. error positions” $= [1,\ 2,\ 7,\ 8]$

$$\textit{Error Value System}: \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & \alpha^2 & \alpha^2 \\ 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & \alpha & \alpha \\ 0 & 0 & 1 & \alpha \end{bmatrix} * e = \begin{bmatrix} 1 \\ \alpha^2 \\ \alpha^2 \\ \alpha \\ \alpha \end{bmatrix}$$

$$e = [0,\ 0,\ 0,\ 1]$$

$$c = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]$$

$$[1,\ 1,\ 1,\ 1]$$

See Also:

GoppaDecode, GoppaEncode, GoppaPrepareSa.

Function: InPolynomial

Convert a polynomial from alpha-form to Root-Of form.

Calling Sequence:

$P :=$ InPolynomial(p, F)
$P :=$ InPolynomial$(p, F, alpha)$

Parameters:

p : `polynomial`
A polynomial over the finite field F.
F : `casa/finite/field`
The finite field of the coefficients of p.

Result:

P : `casa/finite/Polynomial`
A polynomial over a finite field in Root-Of Form.

Description:

Puts a polynomial into Root-Of form
The polynomial is stored in RootOfs.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x+alpha,F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

See Also:

finiteField, RootOf.

Function: NormalPolynomial

Calculate the "Normal Form" of a polynomial

Calling Sequence:

$P :=$ NormalPolynomial(p, F)

Parameters:

p : `expression`
An expression over the finite field F in Root-of Form.
F : `casa/finite/field`
The finite field of the coefficients of p.

Result:

P : `polynomial`
p expanded and collected in alpha-Form.

Description:

Calculates alpha-Form of Root-of Form of a Polynomial.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x+alpha,F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> r := PolynomialRoots(P,F8);
```

$$\begin{aligned}
&r := [[\%1^2 + \%1,\, 0],\, [\%1^2 + \%1,\, 1],\, [\%1^2 + \%1,\, \%1],\\
&[\%1^2 + \%1,\, \%1^2],\, [\%1^2 + \%1,\, \%1^2 + 1],\\
&[\%1^2 + \%1,\, \%1^2 + \%1 + 1],\, [\%1^2 + \%1,\, \%1 + 1],\\
&[\%1^2 + \%1,\, \%1^2 + \%1]]\\
&\%1 := \mathrm{RootOf}(_Z^3 + _Z^2 + 1)
\end{aligned}$$

```
> p := NormalPolynomial(P,F8);
```

$$p := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

See Also:

finiteField, InPolynomial.

Function: OutPolynomial

Calculate the alpha form of a polynomial.

Calling Sequence:

$P :=$ OutPolynomial(p, F)

Parameters:

p : `polynom`
The polynomial over the finite field in Root-of Form.

F : `casa/finite/field`
The finite field of the coefficients of p.

Result:

P : `polynomial`
p in alpha-Form.

Description:

Calculates alpha-Form of Root-of Form of a polynomial.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x+alpha,F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> r := map(F8["mapOut"],PolynomialRoots(P,F8));
```

$$r := [[\alpha^6, 0], [\alpha^6, 1], [\alpha^6, \alpha], [\alpha^6, \alpha^2], [\alpha^6, \alpha^3], [\alpha^6, \alpha^4], [\alpha^6, \alpha^5], [\alpha^6, \alpha^6]]$$

```
> p := OutPolynomial(P,F8);
```

$$p := \alpha^2\, x + \alpha$$

See Also:

finiteField, InPolynomial.

Function: PolynomialRoots

Calculate the roots of a polynomial.

Calling Sequence:

$R :=$ PolynomialRoots(p, F)

Parameters:

p : `polynom`
The polynomial over the finite field F.

F : `casa/finite/field`
The finite field of the coefficients of p.

Result:

R : `list`
The roots of the polynomial p over F.

Description:

Calculates roots of a polynomial in Root-of form.
The polynomial is stored in RootOfs.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x+alpha,F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> r := map(F8["mapOut"],PolynomialRoots(P,F8));
```

$$r := [[\alpha^6, 0], [\alpha^6, 1], [\alpha^6, \alpha], [\alpha^6, \alpha^2], [\alpha^6, \alpha^3], [\alpha^6, \alpha^4], [\alpha^6, \alpha^5], [\alpha^6, \alpha^6]]$$

See Also:

finiteField, InPolynomial.

Function: SubsPolynomial

Evaluate a polynomial at a point in Root-of Form.

Calling Sequence:

$a :=$ SubsPolynomial$(pointp, F)$

Parameters:

point : `list`
  point where to evaluate the polynomial
F : `casa/finite/field`
  The finite field of the coefficients of p.

Result:

a : `root-of expression`
  The value of p at *point*.

Description:

Evaluates a polynomial at a point.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x+alpha,F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> r := PolynomialRoots(P,F8);
```

$$\begin{aligned} r := &[[\%1^2 + \%1,\, 0],\, [\%1^2 + \%1,\, 1],\, [\%1^2 + \%1,\, \%1],\\ &[\%1^2 + \%1,\, \%1^2],\, [\%1^2 + \%1,\, \%1^2 + 1],\\ &[\%1^2 + \%1,\, \%1^2 + \%1 + 1],\, [\%1^2 + \%1,\, \%1 + 1],\\ &[\%1^2 + \%1,\, \%1^2 + \%1]]\\ &\%1 := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)\end{aligned}$$

```
> p := NormalPolynomial(P,F8);
```

$$p := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> a:=  SubsPolynomial([alpha,alpha^2],p,F8);
```

$$a := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2 + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1) + 1$$

See Also:

finiteField, InPolynomial.

Function: mapOutPolynomial

Convert a list of polynomial to alpha form.

Calling Sequence:

$P := \mathsf{mapOutPolynomial}(L, F)$

Parameters:

L : `list of polynomials`
list of polynomials in Root of form.
F : `casa/finite/field`
The finite field of the coefficients of p.

Result:

P : `list of polynomials`
the list of polynomials in alpha form.

Description:

Converts a list of polynomials to a list of polynomials in alpha form.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x + alpha, F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> R := PolynomialRoots(P, F8);
```

$$R := [[\%1^2 + \%1,\, 0],\, [\%1^2 + \%1,\, 1],\, [\%1^2 + \%1,\, \%1],$$
$$[\%1^2 + \%1,\, \%1^2],\, [\%1^2 + \%1,\, \%1^2 + 1],$$
$$[\%1^2 + \%1,\, \%1^2 + \%1 + 1],\, [\%1^2 + \%1,\, \%1 + 1],$$
$$[\%1^2 + \%1,\, \%1^2 + \%1]]$$
$$\%1 := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> Q := InPolynomial(alpha*x + alpha^2, F8);
```

$$Q := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2$$

```
> P:=  mapOutPolynomial([P, Q], F8);
```

$$P := [\alpha^2\, x + \alpha,\, \alpha\, x + \alpha^2]$$

See Also:

finiteField, InPolynomial.

Function: mapSubsPolynomial

Evaluate a polynomial on a list of points.

Calling Sequence:

$P := \mathsf{mapSubsPolynomial}(L, p, F)$

Parameters:

L : `list of points`
list of points on which to eval p.

F : `casa/finite/field`
The finite field of the coefficients of p.
p : `casa/finite/Polynomial`
the polynomial.

Result:

P : `list of root-of expression`
p(P[1),.....]

Description:

Evals a polynomial on a list of points.

Examples:

```
> F8:= finiteField(8);
```

$$F8 := Field$$

```
> P := InPolynomial(alpha^2*x + alpha, F8);
```

$$P := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)$$

```
> R := PolynomialRoots(P ,F8);
```

$$\begin{aligned}R := [&[\%1^2 + \%1,\ 0],\ [\%1^2 + \%1,\ 1],\ [\%1^2 + \%1,\ \%1],\\ &[\%1^2 + \%1,\ \%1^2],\ [\%1^2 + \%1,\ \%1^2 + 1],\\ &[\%1^2 + \%1,\ \%1^2 + \%1 + 1],\ [\%1^2 + \%1,\ \%1 + 1],\\ &[\%1^2 + \%1,\ \%1^2 + \%1]]\\ \%1 :=\ &\mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)\end{aligned}$$

```
> Q := InPolynomial(alpha*x + alpha^2, F8);
```

$$Q := \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)\, x + \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)^2$$

```
> P:=  mapSubsPolynomial([[alpha, alpha^2], [1, alpha]], P, F8);
```

$$\begin{aligned}P &:= [\%1^2 + \%1 + 1,\ \%1^2 + \%1]\\ \%1 &:= \mathrm{RootOf}(\_Z^3 + \_Z^2 + 1)\end{aligned}$$

See Also:

finiteField, InPolynomial.

5.13 Auxiliary Function

Function: delete

Delete an element from a list or a set

Calling Sequence:

$lt :=$ delete(e, l)
$lt :=$ delete$(e, l, "all")$

Parameters:

e : `expression`
  expression to be deleted
l : `list or set`
  a list or a set.

Result:

lt : `list or set`
  a list or a set

Description:

Given a list l the function removes some or all occurences of the expression e.
If "all" is specified as the third argument, all occurences of e are removed from l. Otherwise only the first occurence is deleted.
The result is a sublist of l with some or all occurences of e removed.

Examples:

```
> delete(x,[x,y,z,x]);
```

$$[y, z, x]$$

```
> delete(x,[x,y,z,x],"all");
```

$$[y, z]$$

See Also:

equalProjectivePoints, homogeneousForm, leadingForm, numberOfTerms, pointInAlgSet, setRandomParameters.

Function: equalProjectivePoints

Compare two projective points.

Calling Sequence:

$b :=$ equalProjectivePoints($point1, point2$)

Parameters:

$point1$: `projPoint`
  a first point in projective space
$point2$: `projPoint`
  a second point in projective space

Result:

b : `boolean`
  a boolean value

Description:

This procedure checks whether two given points in a projective space are equal or not.

Examples:

```
> equalProjectivePoints([1,0,0],[0,1,0]);
```

$$false$$

```
> equalProjectivePoints([1,0,2],[3,0,6]);
```

$$true$$

```
> equalProjectivePoints([1,0,0],[1,0,0]);
```

$$true$$

See Also:

.

Function: homogeneousForm

Collects the terms of a polynomial of a certain degree.

Calling Sequence:

$f :=$ homogeneousForm($poly, vars, degree$)

Parameters:

poly : `polynom(anything,`*vars*`)`
A polynomial in the variables vars over some coefficient domain.
vars : `list(name)`
A list of variable names.
degree : `integer`
The degree of the terms that should be collected from the polynomial

Result:

p : `homogeneousPolynom(vars)`
A homogeneous polynom over the same coefficient domain as *poly* in the variables *vars* of degree *degree*.

Description:

homogeneousForm(*poly*,*vars*,*degree*) collects the terms of degree *degree* and returns the form (a homogeneous polynomial) that is built from that terms The sum over all homogeneous forms from degree 0 to the degree of *poly* in *vars*, should give *poly*.

Examples:

```
> f := x^2+3*x*y+y;
```

$$f := x^2 + 3\,x\,y + y$$

```
> f2:=homogeneousForm(f,[x,y],2);
```

$$f2 := 3\,x\,y + x^2$$

```
> f1:=homogeneousForm(f,[x,y],1);
```

$$f1 := y$$

```
> f0:=homogeneousForm(f,[x,y],0);
```

$$f0 := 0$$

```
> evalb(f = f2+f1+f0);
```

$$true$$

See Also:

type/homogeneousPolynomial, leadingForm, homogenize.

Function: homogenize

Homogenizes a polynomial.

Calling Sequence:

$p :=$ homogenize$(poly, vars, x)$

Parameters:

$poly$: `polynom(anything.`$vars$`)`
A polynomial in the variables vars over some coefficient domain.

$vars$: `list(name)`
A list of variable names.

x : `name`
The name of a variable, the homogenizing variable.

Result:

p : `polynom`
A polynom over the same coefficient domain as $poly$ in the variables $vars$ and x.

Description:

homogenize($poly$,$vars$,x) homogenizes a polynomial in $vars$ with respect to the homogenizing variable x.
It is not checked whether the homogenizing variable already appears in the variables $vars$ or in the coeeficient domain. In such a case the result is the same as homogenizing with a variable not appearing in $vars$ or the coefficient domain and then replacing that variable with x. In this case the result might not be homogeneous.

Examples:

```
> homogenize(x^2+3*x^2*y+y,[x,y],z);
```

$$z\,x^2 + 3\,x^2\,y + y\,z^2$$

```
> homogenize(x*y*z+5*y+11*z,[x,y],w);
```

$$x\,y\,z + 5\,w\,y + 11\,w^2\,z$$

```
> homogenize(x*y*z+5*y+11*z,[x,y],x);
```

$$x\,y\,z + 5\,x\,y + 11\,z\,x^2$$

```
> homogenize(x*y*z+5*y+11*z,[x,y],y);
```

$$x\,y\,z + 5\,y^2 + 11\,y^2\,z$$

```
> homogenize(x*y*z+5*y+11*z,[x,y],z);
```

$$x\,y\,z + 5\,z\,y + 11\,z^3$$

See Also:

homogeneousForm, leadingForm, numberOfTerms.

Function: leadingForm

Collects the terms of a polynomial with a degree equal to the degree of the polynomial.

Calling Sequence:

$f :=$ leadingForm($poly, vars$)

Parameters:

poly : `polynom(anything,`$vars$`)`
A polynomial in the variables vars over some coefficient domain.
vars : `list(name)`
A list of variable names.

Result:

p : `homogeneousPolynom(vars)`
A homogeneous polynom over the same coefficient domain as *poly* in the variables *vars* of degree equal to the degree of *poly* in the variables *vars*.

Description:

leadingForm(*poly*,*vars*) collects the terms of highest degree of the polynomial *poly*.
The leading form of the zero polynomial is defined to be zero.

Examples:

```
> f := x^2+3*x*y+y;
```

$$f := x^2 + 3\,x\,y + y$$

```
> g:=homogeneousForm(f,[x,y],3);
```

$$g := 0$$

```
> h:=leadingForm(f,[x,y]);
```

$$h := 3\,x\,y + x^2$$

See Also:

homogeneousForm, homogenize.

Function: numberOfTerms

Determine the number of terms of a polynomial.

Calling Sequence:

$n :=$ numberOfTerms($poly, vars$)

Parameters:

$poly$: `polynom`
A polynomial.
$vars$: `list(name)`
A list of indeterminates.

Result:

n : `integer`
The number of terms of the polynomial $poly$ considered as a polynomial in the indeterminates $vars$.

Description:

The function determines the number of terms of a polynomial $poly$ considered as a polynomial in the indeterminates $vars$.

Examples:

```
> numberOfTerms(0,[x,y]);
                                        1
> numberOfTerms(x,[x,y]);
                                        1
> numberOfTerms(x*y,[x]);
                                        1
> numberOfTerms(x^2+y,[x,y]);
                                        2
> numberOfTerms(x^2+y,[z]);
                                        1
```

See Also:

equalProjectivePoints, homogeneousForm, leadingForm, delete, pointInAlgSet, setRandomParameters.

Function: variableDifferentFrom

Return an unassigned variable that is different from the variables given as arguments.

Calling Sequence:

$v :=$ variableDifferentFrom$(excludedVariables)$

Parameters:

$excludedVariables$: `list(anything)`
A list of names that will not be returned.

Result:

v : `name`
An unevaluated user variable, i.e., a global variable.

Description:

The function returns a variable that is not among the given list of names. The function tries to take the first unevaluated variables from the list provided by the variable $casaDefaultVariables$. If it cannot find an appropriate variable, it takes the first of the variables x1, x2, etc. that has no value.
The function is NOT intended to be used at the user level (though it would not do any harm), but instead serves as a subroutine to create new internal CASA variables.

Examples:

```
> casaDefaultVariables := [x,y,z,u];
```

$$casaDefaultVariables := [x, y, z, u]$$

```
> x:=0; x1:=1;
```

$$x := 0$$
$$x1 := 1$$

```
> excl := [y,u];
```

$$excl := [y, u]$$

```
> excl := [op(excl),variableDifferentFrom(excl)];
```

$$excl := [y, u, z]$$

```
> excl := [op(excl),variableDifferentFrom(excl)];
```

$$excl := [y, u, z, x2]$$

See Also:

.

Index

Bibliography

[CAS94] CASA 2.1: A Maple package for algebraic geometry. CAN Newsbrief 12, http://www.can.nl/CA-Documents/CANnewsletter, March 1994.

[CKM97] S. Collart, M. Kalkbrener, and D. Mall. The Gröbner walk. *Journal of Symbolic Computation*, 24:465–469, 1997.

[Dei99] Harald Deischinger. Algebraic geometric codes. RISC Linz Report Series 99-22, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, August 1999.

[GKWW91] R. Gebauer, M. Kalkbrener, B. Wall, and F. Winkler. CASA: A computer algebra package for constructive algebraic geometry. In Stephen M. Watt, editor, *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 403–410, Bonn, FRG, July 1991. ACM Press.

[GTZ88] Patrizia Gianni, Barry Trager, and Gail Zacharias. Gröbner bases and primary decomposition of polynomial ideals. *Journal of Symbolic Computation*, 6(2–3):149–167, October–December 1988. Computational aspects of commutative algebra.

[Hac00] Michaela Hackl. Multivariate polynomial codes. RISC Linz Report Series 00-05, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, February 2000.

[Hem99] Ralf Hemmecke. CASA — A Maple package to investigate algebraic curves, December 1999. Talk at the University of Duisburg (Germany).

[HHSW98] Ralf Hemmecke, Erik Hillgarter, Wolfgang Schreiner, and Franz Winkler. An evaluation of the state of the CASA system. RISC Linz Report Series 98-16, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, October 1998.

[HHW00] Ralf Hemmecke, Erik Hillgarter, and Franz Winkler. The CASA system. In J. Grabmeier, E. Kaltofen, and V. Weispfenning, editors, *Handbook of Computer Algebra: Foundation, Applications, Systems*. Springer-Verlag, Heidelberg, 2000.

[HW00] Ralf Hemmecke and Franz Winkler. CASA user manual (version casa 2.5). RISC Linz Report Series 00-30, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, September 2000.

[Mňu95] M. Mňuk. *Algebraic and Geometric Approach to Parameterization of Rational Curves.* Phd thesis, Research Institute for Symbolic Computation, Linz, Austria, 1995.

[MW96] M. Mňuk and F. Winkler. CASA—a system for computer aided constructive algebraic geometry. In Jaques Calmet and Carla Limongelli, editors, *Proceedings of the International Symposium DISCO'96*, number 1128 in LNCS, pages 297–307. Springer, 1996.

[MWW93] M. Mňuk, B. Wall, and F. Winkler. CASA reference manual (version 2.1). RISC Linz Report Series 93-63, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, 1993.

[MWW95] M. Mňuk, B. Wall, and F. Winkler. CASA reference manual (version 2.2). RISC Linz Report Series 95-05, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, 1995.

[Sen94] J. R. Sendra. Manipulacion simbolica de curvas en CASA, 1994. Spanish educational conference.

[SW97] J. R. Sendra and F. Winkler. Parametrization of algebraic curves over optimal field extensions. *Journal of Symbolic Computation*, 23:191–207, 1997.

[Tra95a] Q.-N. Tran. A hybrid symbolic-numerical method for tracing surface-to-surface intersections. In A. H. M. Levelt, editor, *The Proceedings of ISSAC'95*, pages 51–58. ACM Press, 1995.

[Tra95b] Q.-N. Tran. On the symbolic-numerical methods for finding the roots of an arbitrary system of nonlinear algebraic equations. In *Proceedings of the First Asian Technology Conference in Mathematics, ATCM-95*, Singapore, 1995.

[Tra96] Q.-N. Tran. *A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization.* Phd thesis, Research Institute for Symbolic Computation (RISC-Linz), University of Linz, Austria, 1996.

[TW97a] Q.-N. Tran and F. Winkler. An overview of CASA—a system for computational algebra and constructive algebraic geometry. In T. V. Effelterre, T. Recio, and F. Winkler, editors, *The Symbolic and Algebraic Computation (SAC) Newsletter*, volume 2, pages 63–72. Stichting CAN, Computer Algebra Nederland, Universiteit van Amsterdam, the Netherlands, 1997.

[TW97b] Quoc-Nam Tran and Franz Winkler. CASA reference manual (version 2.3). RISC Linz Report Series 97-33, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, October 1997.

[Wal78] R. J. Walker. *Algebraic Curves.* Springer-Verlag, second edition, 1978.

[Wal88] B. Wall. Computation of syzygies solution of linear systems over a multivariate polynomial ring. RISC Linz Report Series 88-86, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, 1988.

[Wal91] B. Wall. CASA: Computer algebra software for constructive algebraic geometry in Maple—A primer. RISC Linz Report Series 91-34, Research Institute for Symbolic Computation, 4040 Linz, Austria, Europe, 1991.

[Wal93] B. Wall. *Symbolic Computation with Algebraic Sets.* Phd thesis, Research Institute for Symbolic Computation (RISC-Linz), University of Linz, Austria, 1993.

[Win96] F. Winkler. Algebraic computation in geometry. *Special Issue of J. Math. in Comp. Simulation*, 42(4-6):529–537, 1996.

[Win98] Franz Winkler. CASA — a system for computational algebra and constructive algebraic geometry, 1998. Talk at workshop "Automated Deduction in Geometry", Beijing.