Studying and solving polynomial systems with the RegularChains library

The RegularChains Developer Team
October, 2021

1. Introduction

The RegularChains library offers a variety of commands for solving polynomial systems symbolically and studying their solutions. The input systems may contain polynomial equations $p = 0$, polynomial inequations $p \neq 0$, and polynomial inequalities $p > 0$ or $p \geq 0$. All of those may be nonlinear. For a given system, the coefficients of the input polynomials may be rational numbers, integers modulo a prime, or polynomials depending on parameters. The solution sets computed by the RegularChains library are described by lists of components. Geometrically, such a component can be a point (or a set of points), a curve (or a set of curves), a surface (or a set of surfaces), etc. Computationally, components are represented by a special kind of polynomial system with a triangular shape and other algebraic properties. Depending if the input system consists of equations only, has inequations but no inequalities, or possesses inequalities, the system representing a component is called a *regular chain*, a *regular system* or a *regular semi-algebraic system*, respectively. The word "regular" refers to the interesting algebraic properties that these systems have. The RegularChains library provides types for these different kinds of polynomial systems; this important feature will be illustrated hereafter.

Another design feature of the RegularChains library is the organization of its 135 commands into 7 modules. The top-level of the library gathers the most commonly used commands, whereas the six submodules are dedicated to special topics. The submodules ChainTools, ConstructibleSetTools, and SemiAlgebraicSetTools deal respectively with specific operations on regular chains, regular systems, and regular semi-algebraic systems. The submodule MatrixTools allows the user to handle linear systems over domains with zero-divisors. This is a fundamental tool in the theory of regular chains with many potential applications.

The submodule ParametricSystemTools is devoted to solving systems with parameters, including real root classification and complex root classification of such systems, as demonstrated below. Finally, the submodule FastArithmeticTools provides highly optimized implementation of some fundamental operations on regular chains, similar to some commands of the top-level module and ChainTools. However, the commands of this submodule can only be used under some assumptions and their usage requires care and advanced knowledge.

```
> restart;
```

```
with(RegularChains);
with(ChainTools);
with(MatrixTools);
with(ConstructibleSetTools);
with(ParametricSystemTools);
with(SemiAlgebraicSetTools);
with(FastArithmeticTools);
with(AlgebraicGeometryTools);
```

[*AlgebraicGeometryTools*, *ChainTools*, *ConstructibleSetTools*, *Display*, *DisplayPolynomialRing*, *Equations*, *ExtendedRegularGcd*, *FastArithmeticTools*, *Inequations*, *Info*, *Initial*, *Intersect*, *Inverse*, *IsRegular*, *LazyRealTriangularize*, *MainDegree*, *MainVariable*, *MatrixCombine*, *MatrixTools*, *NormalForm*, *ParametricSystemTools*, *PolynomialRing*, *Rank*, *RealTriangularize*, *RegularGcd*, *RegularizeInitial*, *SamplePoints*, *SemiAlgebraicSetTools*, *Separant*, *SparsePseudoRemainder*, *SuggestVariableOrder*, *TRDFM_elim_eqsfirst*, *TRDconvex_union*, *Tail*, *Triangularize*]

[*Chain*, *ChangeOfCoordinates*, *ChangeOfOrder*, *Construct*, *Cut*, *DahanSchostTransform*, *Dimension*, *Empty*, *EqualSaturatedIdeals*, *EquiprojectableDecomposition*, *Extend*, *ExtendedNormalizedGcd*, *IsAlgebraic*, *IsEmptyChain*, *IsInRadical*, *IsInSaturate*, *IsIncluded*, *IsPrimitive*, *IsStronglyNormalized*, *IsZeroDimensional*, *IteratedResultant*, *LastSubresultant*, *Lift*, *ListConstruct*, *NormalizeRegularChain*, *NumberOfSolutions*, *Polynomial*, *Regularize*, *RemoveRedundantComponents*, *SeparateSolutions*, *Squarefree*, *SquarefreeFactorization*, *SubresultantChain*, *SubresultantOfIndex*, *Under*, *Upper*]

[*IsZeroMatrix*, *JacobianMatrix*, *LowerEchelonForm*, *MatrixInverse*, *MatrixMultiply*, *MatrixOverChain*]

[*Complement*, *ConstructibleSet*, *CylindricalDecompose*, *Difference*, *EmptyConstructibleSet*, *GeneralConstruct*, *Intersection*, *IsContained*, *IsEmpty*, *MakePairwiseDisjoint*, *PolynomialMapImage*, *PolynomialMapPreimage*, *Projection*, *QuasiComponent*, *RationalMapImage*, *RationalMapPreimage*, *RefiningPartition*, *RegularSystem*, *RegularSystemDifference*, *RepresentingChain*, *RepresentingInequations*, *RepresentingRegularSystems*, *SeparateZeros*, *Union*]

[*BelongsTo*, *BorderPolynomial*, *CoefficientsInParameters*, *ComplexRootClassification*, *ComprehensiveTriangularize*, *DefiningSet*, *DiscriminantSequence*, *DiscriminantSet*, *PreComprehensiveTriangularize*, *RealComprehensiveTriangularize*, *RealRootClassification*, *Specialize*]

[*BoxValues*, *Complement*, *CylindricalAlgebraicDecompose*, *Difference*, *DisplayParametricBox*, *DisplayQuantifierFreeFormula*, *EmptySemiAlgebraicSet*, *Intersection*, *IsContained*, *IsEmpty*, *IsParametricBox*, *LinearSolve*, *PartialCylindricalAlgebraicDecomposition*, *PositiveInequalities*, *Projection*, *QuantifierElimination*, *RealRootCounting*, *RealRootIsolate*,

RefineBox, RefineListBox, RemoveRedundantComponents, RepresentingBox, RepresentingChain, RepresentingQuantifierFreeFormula, RepresentingRootIndex, SignAtBox, VariableOrdering]
[*BivariateModularTriangularize, IteratedResultantDim0, IteratedResultantDim1, NormalFormDim0, NormalizePolynomialDim0, NormalizeRegularChainDim0, RandomRegularChainDim0, RandomRegularChainDim1, ReduceCoefficientsDim0, RegularGcdBySpecializationCube, RegularizeDim0, ResultantBySpecializationCube, SubresultantChainSpecializationCube*]
[*Cylindrify, IntersectionMultiplicity, IsTransverse, LimitPoints, RationalFunctionLimit, RegularChainBranches, TangentCone, TangentPlane, TriangularizeWithMultiplicity*] **(1.1)**

2. Polynomial systems and regular chains

This section introduces the user to the top-level module of the RegularChains library. The concept of a regular chain is based on a recursive and univariate vision of a polynomial. Several commands manipulating polynomials in this manner are illustrated below. The most commonly used command of the library is called Triangularize and is illustrated by several examples. It takes a polynomial system as input and returns a description of its solution set. If the input system consists of equations only, then this description is a list of regular chains. The case where the input system admits inequations can be handled by Triangularize or by the command GeneralConstruct of the ConstructibleSetTools submodule. In case of inequalities, and more generally for computing the real solutions of a polynomial system, the commands RealTriangularize, LazyRealTriangularize and SamplePoints of the top-level module can be used. Different commands are also available in the modules ParametricSystemTools and SemiAlgebraicSetTools for the same purpose of real solving but for particular types of systems: they will be presented later in this document.

2.1 A univariate vision of polynomials

Define the polynomial ring $\mathbb{Q}[x, y, z]$ with the variable ordering $x > y > z$. The field $\mathbb{Q}$ of rational numbers is the default coefficient ring.

```
> R := PolynomialRing([x, y, z]);
```

$$R := polynomial\_ring \quad \textbf{(2.1.1)}$$

The internal representation of this polynomial ring can be accessed as follows.

```
> Display(R);
```

$$\begin{cases} Variables & : & [x, y, z] \\ Parameters & : & \varnothing \\ Characteristic & : & 0 \end{cases} \quad \textbf{(2.1.2)}$$

Consider the following polynomial.

```
> p := (y+1)*x^3+(z+4)*x+3;
```

$$p := (y+1)\,x^3 + (z+4)\,x + 3 \quad \textbf{(2.1.3)}$$

The *main variable* of p as a polynomial of R is given by:

```
> MainVariable(p, R);
```

$$x \tag{2.1.4}$$

The *initial* (or *leading coefficient*) of p:

```
> Initial(p, R);
```

$$y + 1 \tag{2.1.5}$$

The degree of p in the main variable:

```
> MainDegree(p, R);
```

$$3 \tag{2.1.6}$$

The rank of p, which is the main variable of p raised to the main degree of p:

```
> Rank(p, R);
```

$$x^3 \tag{2.1.7}$$

The tail of p.

```
> Tail(p, R);
```

$$x z + 4 x + 3 \tag{2.1.8}$$

The separant of p, which is the derivative of p with regards to its main variable:

```
> Separant(p, R);
```

$$3 x^2 y + 3 x^2 + z + 4 \tag{2.1.9}$$

Change the ordering to $z > y > x$:

```
> R := PolynomialRing([z, y, x]);
  Display(R);
  p := expand((y+1)*x^3+(z+4)*x+3);
  MainVariable(p, R);
  Initial(p, R);
  Rank(p, R);
  Tail(p, R);
  Separant(p, R);
```

$$R := polynomial\_ring$$

$$\begin{cases} Variables & : & [z, y, x] \\ Parameters & : & \varnothing \\ Characteristic & : & 0 \end{cases}$$

$$p := x^3 y + x^3 + x z + 4 x + 3$$

$$z$$

$$x$$

$$z$$

$$x^3 y + x^3 + 4 x + 3$$

$$x \tag{2.1.10}$$

Consider the polynomial ring $\mathbb{Z}_3[z, y, x]$ with the ordering $z > y > x$.

```
> R := PolynomialRing([z, y, x], 3);
  Display(R);
  Initial(p, R);
  Tail(p, R);
```

$$R := polynomial\_ring$$

$$\begin{cases} \text{Variables} & : & [z, y, x] \\ \text{Parameters} & : & \varnothing \\ \text{Characteristic} & : & 3 \end{cases}$$

$$x^3 y + x^3 + x \qquad (2.1.11)$$

2.2 Computing the subresultants of two polynomials

The command SubresultantChain in the ChainTools module is used to compute the subresultant chain of two polynomials. Let us illustrate this by an example.

Define a ring of polynomials.

```
> R := PolynomialRing([y, x]);
```

$$R := \text{polynomial\_ring} \qquad (2.2.1)$$

Define two polynomials of **R**.

```
> f1 := (y^2+6)*(x-1)-y*(x^2+1);
```

$$f1 := (y^2+6)(x-1) - y(x^2+1) \qquad (2.2.2)$$

```
> f2 := (x^2+6)*(y-1)-x*(y^2+1);
```

$$f2 := (x^2+6)(y-1) - x(y^2+1) \qquad (2.2.3)$$

Compute their subresultant chain w.r.t the variable y

```
> src := SubresultantChain(f1, f2, y, R);
```

$$src := \text{subresultant\_chain} \qquad (2.2.4)$$

The result is of type *subresultant_chain* and encodes all the polynomials in the subresultant chain. To display the i_{th} subresulant polynomial, one can call the command SubresultantOfIndex. For example, to know the subresultant of index 0, that is the resultant, one can do as follows

```
> res := SubresultantOfIndex(0, src, R);
```

$$res := 2x^6 - 22x^5 + 102x^4 - 274x^3 + 488x^2 - 552x + 288 \qquad (2.2.5)$$

To display all the subresultant polynomials, one can call the command Display.

```
> Display(src, R);
```

$$\begin{cases} (-x^2 + 5x - 6)y - x^3 + 6x^2 - 11x + 6 \\ 2x^6 - 22x^5 + 102x^4 - 274x^3 + 488x^2 - 552x + 288 \end{cases} \qquad (2.2.6)$$

The encoding of the subresultants can be controlled by an optional argument. This encoding can be *by values*, over the *monomial basis* or via the *Bezout Matrix*. When this optional argument is not specified, the encoding is chosen according to a heuristical algorithm.

```
> src := SubresultantChain(f1, f2, y, R, representation =
  BezoutMatrix); op(src);
```

$$src := subresultant\_chain$$

$$\begin{aligned}\mathrm{table}\Big(\Big[&SRC\_MDEG = 1,\ SRC\_MVAR = y,\ type = subresultant\_chain,\\ &SRC\_POLYQ = x^2y - xy^2 - x^2 - x + 6y - 6,\ SRC\_POLYP = -x^2y + xy^2\\ &- y^2 + 6x - y - 6,\ SRC\_MATRIX\\ &= \begin{bmatrix} -x^2 + 5x - 6 & -x^3 + 6x^2 - 11x + 6 \\ -x^3 + 6x^2 - 11x + 6 & x^4 - 5x^3 + 13x^2 - 35x + 42 \end{bmatrix},\ representation\\ &= BezoutMatrix\Big]\Big)\end{aligned} \tag{2.2.7}$$

```
> src := SubresultantChain(f1, f2, y, R, representation =
  MonomialBasis); op(src);
```

$$src := subresultant\_chain$$

$$\begin{aligned}\mathrm{table}\big([&SRC\_MDEG = 1,\ SRC\_MVAR = y,\ type = subresultant\_chain,\\ &SRC\_POLYQ = x^2y - xy^2 - x^2 - x + 6y - 6,\ SRC\_POLYP = -x^2y + xy^2\\ &- y^2 + 6x - y - 6,\ subresultant\_chain\_vector = [2x^6 - 22x^5 + 102x^4\\ &- 274x^3 + 488x^2 - 552x + 288,\ -x^3 - x^2y + 6x^2 + 5xy - 11x - 6y + 6,\\ &x^2y - xy^2 - x^2 - x + 6y - 6,\ -x^2y + xy^2 - y^2 + 6x - y - 6],\\ &representation = MonomialBasis]\big)\end{aligned} \tag{2.2.8}$$

2.3 Solving systems of equations with regular chains

Define a ring of polynomials.

```
> R := PolynomialRing([x, y, z]);
```

$$R := polynomial\_ring \tag{2.3.1}$$

Define a set of polynomials of R.

```
> sys := {x+y+z^2-1, x+y^2+z-1, x^2+y+z-1};
```

(2.3.2)

$$sys := \{z^2 + x + y - 1, y^2 + x + z - 1, x^2 + y + z - 1\} \quad \textbf{(2.3.2)}$$

Ideally, one would like to decompose the solutions of *sys* into a list of points. This is what the command Triangularize does using symbolic expressions. In the output decomposition, some points are grouped because they share some properties. These groups are called *regular chains*.

A regular chain is a system of equations and inequations, satisfying special algebraic properties. First, the equation part consists of non-constant polynomials with pairwise different main variables; thus, the equation part has a triangular shape. Second, the inequation part consists of the initials of the polynomials defining the equations; moreover, the product of those initials is regular (that is, not a zero-divisor) modulo the saturated ideal of the equation part. This technical condition can be skipped by the non-expert reader, since in most practical examples the inequations are trivial and can be ignored. This is the case in our example where all initials are equal to 1, leading to trivial inequations.

```
> l := Triangularize(sys, R);
```

$$l := [regular\_chain, regular\_chain, regular\_chain, regular\_chain] \quad \textbf{(2.3.3)}$$

```
> map(Equations, l, R);
  map(Inequations, l, R);
  map(NumberOfSolutions, l, R);
```

$$[[x - z, y - z, z^2 + 2z - 1], [x, y, z - 1], [x, y - 1, z], [x - 1, y, z]]$$
$$[\varnothing, \varnothing, \varnothing, \varnothing]$$
$$[2, 1, 1, 1] \quad \textbf{(2.3.4)}$$

The result is to be interpreted as follows: The system *sys* is equivalent to a union of four regular chains. The equations for the first such regular chain are $x - 1 = y = z = 0$, the equations for the second regular chain are $x = y - 1 = z = 0$, etc. None of the four regular chains have any inequations. The first three regular chains have exactly one solution, and the last one has two solutions.

Note that you can also specify inequations of the form $p \neq 0$. For example, you can specify that $x - z$ must not vanish.

```
> l := Triangularize(sys, [x-z], R);
  map(Equations, l, R);
  map(NumberOfSolutions, l, R);
```

$$l := [regular\_chain, regular\_chain]$$
$$[[x - 1, y, z], [x, y, z - 1]]$$
$$[1, 1] \quad \textbf{(2.3.5)}$$

2.4 Solving polynomial systems with infinitely many solutions

In the previous examples, the polynomial systems have finitely many solutions. To illustrate how the Triangularize command handles systems with infinitely many solutions, consider the following parametric linear system with unknowns x, y and parameters a, b, c, d, g, h.

```
> R := PolynomialRing([x, y, a, b, c, d, g, h]);
  sys := {a*x+b*y-g, c*x+d*y-h};
```

$$R := polynomial\_ring$$
$$sys := \{a\,x + b\,y - g,\ c\,x + d\,y - h\} \qquad \textbf{(2.4.1)}$$

By default, the Triangularize command computes the *generic solutions* of the input system. For this parametric linear system, this implies its determinant is assumed to be nonzero.

```
> l := Triangularize(sys, R);
  map(Equations, l, R);
  map(Inequations, l, R);
```

$$l := [regular\_chain]$$
$$[[c\,x + y\,d - h,\ (a\,d - b\,c)\,y - a\,h + c\,g]]$$
$$[\{c,\ d\,a - b\,c\}] \qquad \textbf{(2.4.2)}$$

Thus, the resulting regular chain consists of the two equations $c\,x + d\,y - h = (a\,d - b\,c)y + c\,g - a\,h = 0$ and the two inequations $c \neq 0 \neq a\,d - b\,c$, the first of which reflects the choice of c as a pivot.

The other available Maple commands for solving such a system do a similar job.

```
> solve(sys, {x, y});
```

$$\left\{x = -\frac{b\,h - d\,g}{d\,a - b\,c},\ y = \frac{a\,h - c\,g}{d\,a - b\,c}\right\} \qquad \textbf{(2.4.3)}$$

```
> Groebner[Solve](sys, {x, y});
```

$$\{[[d\,x\,a - b\,c\,x + b\,h - d\,g,\ y\,a\,d - y\,b\,c - a\,h + c\,g],\ plex(y, x),\ \varnothing]\} \qquad \textbf{(2.4.4)}$$

The Triangularize command can do more if the option "output=lazard" is used. In this case, all the solutions of the input are computed, generic or not. This implies that the case where the determinant vanishes is also considered and solved. For each regular chain below, its equations and inequations are printed.

```
> l := Triangularize(sys, R, output = lazard):
  seq(Display(l[i], R), i = 1..nops(l));
```

$$\left\{\begin{array}{c} c\,x + d\,y - h = 0 \\ (d\,a - b\,c)\,y - h\,a + c\,g = 0 \\ d\,a - b\,c \neq 0 \\ c \neq 0 \end{array}\right\},\ \left\{\begin{array}{c} c\,x + d\,y - h = 0 \\ d\,a - b\,c = 0 \\ h\,b - d\,g = 0 \\ c \neq 0 \\ d \neq 0 \\ h \neq 0 \end{array}\right\},\ \textbf{(2.4.5)}$$

$$\left\{\begin{array}{c} a\,x + b\,y - g = 0 \\ d\,y - h = 0 \\ c = 0 \\ a \neq 0 \\ d \neq 0 \end{array}\right\},\ \left\{\begin{array}{c} d\,y - h = 0 \\ a = 0 \\ h\,b - d\,g = 0 \\ c = 0 \\ d \neq 0 \\ h \neq 0 \end{array}\right\},\ \left\{\begin{array}{c} c\,x - h = 0 \\ h\,a - c\,g = 0 \\ b = 0 \\ d = 0 \\ c \neq 0 \\ h \neq 0 \end{array}\right\},$$

$$\left\{\begin{array}{c} a\,x+b\,y-g=0 \\ c=0 \\ d=0 \\ h=0 \\ a\neq 0 \end{array}\right\},\left\{\begin{array}{c} c\,x+d\,y=0 \\ d\,a-b\,c=0 \\ g=0 \\ h=0 \\ c\neq 0 \\ d\neq 0 \end{array}\right\},\left\{\begin{array}{c} b\,y-g=0 \\ a=0 \\ c=0 \\ d=0 \\ h=0 \\ b\neq 0 \end{array}\right\},$$

$$\left\{\begin{array}{c} y=0 \\ a=0 \\ c=0 \\ g=0 \\ h=0 \end{array}\right\},\left\{\begin{array}{c} x=0 \\ b=0 \\ d=0 \\ g=0 \\ h=0 \end{array}\right\},\left\{\begin{array}{c} a=0 \\ b=0 \\ c=0 \\ d=0 \\ g=0 \\ h=0 \end{array}\right\}$$

One may also want to "push" some parameters into the field of coefficients, that is, to solve over a field of rational functions. Consider a new polynomial ring over the field $\mathbb{Q}(g,h)$ of rational functions in g and h.

```
> R2 := PolynomialRing([x, y, a, b, c, d], {g, h}) :
  l := Triangularize(sys, R2, output = lazard) :
  seq(Display(li, R2), i = 1..nops(l));
```

$$\left\{\begin{array}{c} c\,x+d\,y-h=0 \\ (d\,a-b\,c)\,y-h\,a+c\,g=0 \\ d\,a-b\,c\neq 0 \\ c\neq 0 \end{array}\right\},\left\{\begin{array}{c} c\,x+d\,y-h=0 \\ d\,a-b\,c=0 \\ h\,b-d\,g=0 \\ c\neq 0 \\ d\neq 0 \end{array}\right\}, \qquad (2.4.6)$$

$$\left\{\begin{array}{c} a\,x+y\,b-g=0 \\ d\,y-h=0 \\ c=0 \\ a\neq 0 \\ d\neq 0 \end{array}\right\},\left\{\begin{array}{c} d\,y-h=0 \\ a=0 \\ h\,b-d\,g=0 \\ c=0 \\ d\neq 0 \end{array}\right\},\left\{\begin{array}{c} c\,x-h=0 \\ h\,a-c\,g=0 \\ b=0 \\ d=0 \\ c\neq 0 \end{array}\right\}$$

Finally, we solve over $\mathbb{Z}_3(g,h)$, that is, over the field of rational functions in g and h modulo 3.

```
> R3 := PolynomialRing([x, y, a, b, c, d], {g, h}, 3);
  l := Triangularize(sys, R3, output = lazard);
```

```
op(Display(l, R3));
```

$$R_3 := polynomial\_ring$$

$$l := [regular\_chain, regular\_chain, regular\_chain, regular\_chain, regular\_chain]$$

$$\left\{ \begin{array}{c} c\,x + d\,y + 2\,h = 0 \\ (d\,a + 2\,b\,c)\,y + 2\,h\,a + c\,g = 0 \\ d\,a + 2\,b\,c \neq 0 \\ c \neq 0 \end{array} \right\}, \left\{ \begin{array}{c} c\,x + d\,y + 2\,h = 0 \\ d\,a + 2\,b\,c = 0 \\ h\,b + 2\,d\,g = 0 \\ c \neq 0 \\ d \neq 0 \end{array} \right\}, \quad \textbf{(2.4.7)}$$

$$\left\{ \begin{array}{c} a\,x + y\,b + 2\,g = 0 \\ d\,y + 2\,h = 0 \\ c = 0 \\ a \neq 0 \\ d \neq 0 \end{array} \right\}, \left\{ \begin{array}{c} d\,y + 2\,h = 0 \\ a = 0 \\ h\,b + 2\,d\,g = 0 \\ c = 0 \\ d \neq 0 \end{array} \right\}, \left\{ \begin{array}{c} c\,x + 2\,h = 0 \\ h\,a + 2\,c\,g = 0 \\ b = 0 \\ d = 0 \\ c \neq 0 \end{array} \right\}$$

2.5 Regular chains and polynomial gcds

The main subroutine of the Triangularize command is the RegularGcd command, which computes polynomial gcds modulo regular chains. This routine is illustrated hereafter together with basic commands on regular chains from the ChainTools submodule.

```
> R := PolynomialRing([y, x]);
```

$$R := polynomial\_ring \quad \textbf{(2.5.1)}$$

Here's the empty regular chain and its internal representation.

```
> Empty(R);
  op(Empty(R));
```

$$regular\_chain$$

$$property, polynomials, type, ModulePrint, \textbf{module}()\ \textbf{option}\ record;\ \textbf{export}\ property, polynomials, type, ModulePrint;\ \textbf{end module} \quad \textbf{(2.5.2)}$$

You can construct a new regular chain by adding a (suitable) polynomial p to an existing regular chain T. This operation may return a list of regular chains. Indeed, checking that the initial of p is regular with regards to (the saturated ideal of) T may split T.

On the example below, this does not happen; the regular chain T is empty.

```
> l := Construct(x^2+1, Empty(R), R);
  rc := l[1];
```

$$l := [regular\_chain]$$

$$rc := regular\_chain \quad \textbf{(2.5.3)}$$

The internal representation of the new regular chain *rc* is as follows. For each polynomial in the regular chain, the data structure stores its main variable, its main degree, and its initial. These quantities are needed quite frequently and are therefore cached to avoid recomputing them each time they are needed.

The internal representation also shows a property about the regular chain: *isPrime*. This means that its saturated ideal is a prime ideal. The knowledge of such property helps to speed up computations.

```
> op(rc);
```

$$regular\_chain \qquad (2.5.4)$$

Now compute the greatest common divisor of two polynomials p_1 and p_2 modulo *rc*. The example is designed such that $y+1$ is a gcd.

```
> p1 := expand(((x+1)·y+x)·(y+1));
  p2 := expand(((x-1)·y+x)·(y+1));
```

$$p_1 := x y^2 + 2 x y + y^2 + x + y$$
$$p_2 := x y^2 + 2 x y - y^2 + x - y \qquad (2.5.5)$$

In general, the output of a gcd computation modulo a regular chain is a list of "cases".

In the above example, no case splits can happen since *rc* defines a field. However, note that the output, below, is not exactly the one expected.

```
> rg := RegularGcd(p1, p2, y, rc, R); factor(rg[1])
```

$$rg := [[2 x y + 2 x, regular\_chain]]$$
$$2 x (y+1) \qquad (2.5.6)$$

However, the extraneous factor $2x$ is a unit modulo *rc*. Thus, the expected and the computed gcds are in fact associate, and therefore the result is correct.

Consider an example where a split is necessary during a gcd computation. To create such an example, you need to make sure that the input equations do not factor, because otherwise the Construct command will already split the system. The example below has three variables; the two variables x and y are used for building a regular chain with two polynomials mx and my.

```
> R := PolynomialRing([z, y, x]);
```

$$R := polynomial\_ring \qquad (2.5.7)$$

The first polynomial is irreducible over $\mathbb{Q}$.

```
> mx := x^2-2;
```

$$mx := x^2 - 2 \qquad (2.5.8)$$

Thus, constructing a regular chain from it produces a single output. Make a copy of it for later use.

```
> rc := Construct(mx, Empty(R), R)[1];
  rcx := rc;
```

$$rc := regular\_chain$$
$$rcx := rc \qquad (2.5.9)$$

The Construct command has detected that (the saturated ideal of) *rc* is a prime ideal.

```
> op(rc);
```

$$regular\_chain \qquad (2.5.10)$$

The second polynomial is not irreducible in $\mathbb{Q}[x][y]$. The Construct command would easily discover this fact and would split it when constructing a regular chain involving rc.

```
> my := expand((y-x)*(y+x-1));
```

$$my := -x^2 + y^2 + x - y \qquad (2.5.11)$$

By reducing my modulo rc, the reduced polynomial becomes irreducible in $\mathbb{Q}[x][y]$.

```
> my := SparsePseudoRemainder(my, rc, R);
```

$$my := y^2 + x - y - 2 \qquad (2.5.12)$$

Extend rc by my.

```
> l := Construct(my, rc, R);rc := l[1];
```

$$l := [regular\_chain]$$
$$rc := regular\_chain \qquad (2.5.13)$$

The Construct command uses inexpensive criteria to detect whether the associated saturated ideal is prime or not. These criteria fail here. So the constructed regular chain rc has no particular properties.

```
> op(rc);Equations(rc, R);
```

$$regular\_chain$$
$$[y^2 - y + x - 2, x^2 - 2] \qquad (2.5.14)$$

Compute the gcd of p_1 and p_2, defined below, modulo rc.

```
> p1 := (y - x) z + (x + y - 1) (z + 1);
  p2 := y - x + (x + y - 1) (z + 1)
```

$$p_1 := (y - x)\, z + (y + x - 1)\, (z + 1)$$
$$p_2 := y - x + (y + x - 1)\, (z + 1) \qquad (2.5.15)$$

The example is constructed in such a way that splitting is needed: modulo the first factor of my, namely $y - x$, the gcd is $z + 1$, while it is constant modulo the second factor, $y + x - 1$. Correspondingly, there are two branches.

```
> rg := RegularGcd(p1, p2, z, rc, R);
```

$$rg := [[(y + x - 1)\, z + 2\, y - 1, regular\_chain], [3\, y^2 + (-2\, x - 2)\, y - x^2 + 2\, x, regular\_chain]] \qquad (2.5.16)$$

Check the first case.

```
> g1, rc1 := op(rg1);
  eq1 := Equations(rc1, R);
  SparsePseudoRemainder((2 x + 1) eq1_1, rcx, R)
```

$$g_1, rc1 := (y + x - 1)\, z + 2\, y - 1, regular\_chain$$
$$eq_1 := [(2\, x - 1)\, y + x - 4, x^2 - 2]$$
$$-7\, x + 7\, y \qquad (2.5.17)$$

Note that rc_1 is not normalized. The SparsePseudoRemainder computation shows that it corresponds to the "modulo $y - x$" branch.

```
> u := SparsePseudoRemainder(g1, rc1, R);
  factor(u)
```

$$u := -4zx - 4x + 9z + 9$$
$$-(z+1)(4x-9) \qquad \textbf{(2.5.18)}$$

This is the desired result, up to a multiplicative factor $-4x+9$ which is a unit modulo rc_1.

Check the second case. Again, the regular chain rc_2 is not normalized, and the SparsePseudoRemainder computation shows that it corresponds to the "modulo $y+x-1$" branch.

```
> g2, rc2 := op(rg2);
  eq2 := Equations(rc2, R);
  SparsePseudoRemainder((2 x + 1) eq2_1, rcx, R)
```

$$g_2, rc2 := 3y^2 + (-2x-2)y - x^2 + 2x, regular\_chain$$
$$eq_2 := [(2x-1)y - 3x + 5, x^2 - 2]$$
$$7x + 7y - 7 \qquad \textbf{(2.5.19)}$$

Check that g_2 is a unit modulo rc_2. First, reduce it modulo rc_2. Then ask if the result is invertible modulo rc_2. The answer proves it is.

```
> h := SparsePseudoRemainder(g2, rc2, R);
  out := Inverse(h, rc2, R);
  ih := out1_1;
  SparsePseudoRemainder(h ih, rc2, R)
```

$$h := -72x + 113$$
$$out := [[[113 + 72x, 2401, regular\_chain]], [\,]]$$
$$ih := 113 + 72x$$
$$2401 \qquad \textbf{(2.5.20)}$$

2.6 Solving systems of equations incrementally

The central command of the RegularChains library is the Triangularize command. The algorithm behind this command solves systems of equations incrementally. To do this, we rely on an important operation Intersect, which computes the common part of a hypersurface and the quasi-component of a regular chain. See the help page of Intersect for a more formal description of it. Let us now illustrate how to use it to solve a polynomial system incrementally.

Define a ring of polynomials.

```
> vars := [x, y, z];R := PolynomialRing(vars);
```

$$vars := [x, y, z]$$
$$R := polynomial\_ring \qquad \textbf{(2.6.1)}$$

Define a set of equations.

```
> sys := [x^2+y+z-1, x+y^2+z-1, x+y+z^2-1];
```

$$sys := [x^2+y+z-1,\ y^2+x+z-1,\ z^2+x+y-1] \qquad (2.6.2)$$

Define the empty regular chain.

```
> rc := Empty(R);
```

$$rc := regular\_chain \qquad (2.6.3)$$

Solve the first equation.

```
> dec := Intersect(sys[1], rc, R);map(Equations, dec, R);
```

$$dec := [regular\_chain]$$

$$[[x^2+y+z-1]] \qquad (2.6.4)$$

Solve the first and second equations.

```
> dec := [seq(op(Intersect(sys[2], rc, R)), `in`(rc, dec))];map
  (Equations, dec, R);
```

$$dec := [regular\_chain, regular\_chain]$$

$$[[x-y,\ y^2+y+z-1],\ [x+y-1,\ y^2-y+z]] \qquad (2.6.5)$$

Solve the three equations together.

```
> dec := [seq(op(Intersect(sys[3], rc, R)), `in`(rc, dec))];Display
  (dec, R);
```

$$dec := [regular\_chain, regular\_chain, regular\_chain, regular\_chain]$$

$$\left[\left\{\begin{array}{l} x-z=0 \\ y-z=0 \\ z^2+2z-1=0 \end{array}\right.,\ \left\{\begin{array}{l} x=0 \\ y=0 \\ z-1=0 \end{array}\right.,\ \left\{\begin{array}{l} x-1=0 \\ y=0 \\ z=0 \end{array}\right.,\ \left\{\begin{array}{l} x=0 \\ y-1=0 \\ z=0 \end{array}\right.\right] \qquad (2.6.6)$$

3. Real solutions of polynomial systems

The RegularChains library offers a variety of tools to compute the real solutions of polynomial systems. A first group of commands (namely RealTriangularize, LazyTriangularize and SamplePoints) deals with arbitrary semi-algebraic systems. That is, given any system S of polynomial equations, polynomial inequations and polynomial inequalities (strict or large) these commands produce information about the real solutions of this system.

RealTriangularize returns a full description of the real solutions of S: it computes simpler systems $S_1,\ldots, S_e$ such that a point is a solution of S if and only if it is a solution of one of the systems $S_1,\ldots, S_e$. Each of these systems

has a triangular shape and remarkable properties: for this reason it is called a regular semi-algebraic system and the set of the $S_1,\ldots, S_e$ is called a full triangular decomposition of S.

LazyRealTriangularize allows the user to compute a triangular decomposition of S in an interactive manner. This feature is particularly well adapted for systems that are hard to solve. For such systems, LazyRealTriangularize returns the components of S of maximum dimension together with unevaluated recursive calls, such that, when fully evaluated, these calls produce the other components of S (which are generally harder to compute).

SamplePoints is even a lazier (and thus much cheaper) way of solving: it produces at least one sample point per connected component of the solution set of S. This way of solving is often sufficient in practical problems.

A second group of commands compute the real solutions of particular types of polynomial systems (such as systems with finitely many complex solutions or parametric systems) or provide advanced features (such as CylindricalAlgebraicDecompose, LinearSolve, Projection, Difference). All these commands except RealRootClassification and RealComprehensiveTriangularize are located in the subpackage SemiAlgebraicSetTools.

3.1 RealTriangularize: solving systems of equations, inequations and inequalities

Consider the generic equation of degree two.

```
> R := PolynomialRing([x, c, b, a]);
  sys := [a*x^2+b*x+c=0];
```

$$R := polynomial\_ring$$
$$sys := [a\,x^2 + b\,x + c = 0] \qquad \textbf{(3.1.1)}$$

Compute a triangular decomposition of the 4-variable hypersurface it defines.

```
> dec := RealTriangularize(sys, R);
```

$$dec := [regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system] \qquad \textbf{(3.1.2)}$$

```
> Display(dec, R);
```

$$\left\{ \begin{array}{l} a\,x^2 + b\,x + c = 0 \\ -4\,c\,a + b^2 > 0 \textbf{ and } a \neq 0 \end{array} \right. , \left\{ \begin{array}{l} 2\,a\,x + b = 0 \\ 4\,a\,c - b^2 = 0 \\ a \neq 0 \end{array} \right. , \left\{ \begin{array}{l} b\,x + c = 0 \\ a = 0 \\ b \neq 0 \end{array} \right. , \qquad \textbf{(3.1.3)}$$

$$\left[\begin{cases} c = 0 \\ b = 0 \\ a = 0 \end{cases}\right]$$

Consider the record output format.

```
> RealTriangularize(sys, R, output=record);
```

$$\left[\begin{cases} a x^2 + b x + c = 0 \\ -4 c a + b^2 > 0 \\ a \neq 0 \end{cases}, \begin{cases} 2 a x + b = 0 \\ 4 a c - b^2 = 0 \\ a \neq 0 \end{cases}, \begin{cases} b x + c = 0 \\ b \neq 0 \\ a = 0 \end{cases}, \begin{cases} c = 0 \\ b = 0 \\ a = 0 \end{cases}\right] \quad \textbf{(3.1.4)}$$

Next, we consider a system of equations, inequations and inequalities.

```
> R := PolynomialRing([y, x, b, a]);
```

$$R := polynomial\_ring \quad \textbf{(3.1.5)}$$

```
> sys := [x^3-3*y^2*x+a*x+b=0, 3*x^2-y^2+a=0, 1-y*x>0, y<>0];
```

$$sys := [x^3 - 3 y^2 x + a x + b = 0, 3 x^2 - y^2 + a = 0, 0 < -y x + 1, y \neq 0] \quad \textbf{(3.1.6)}$$

```
> out := RealTriangularize(sys, R);
```

$$out := [regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system] \quad \textbf{(3.1.7)}$$

```
> Display(out, R);
```

$$\left[\left\{\begin{array}{c} y^2 - 3 x^2 - a = 0 \\ 8 x^3 + 2 a x - b = 0 \\ -y x + 1 > 0 \\ 4 a^3 + 27 b^2 > 0 \textbf{ and } 4 b^2 a^3 - 16 a^4 + 27 b^4 - 512 a^2 - 4096 \neq 0 \end{array}\right\}, \left\{\begin{array}{c} x y + 1 = 0 \\ (2 a^3 + 18 b^2 + 32 a) x + (-a^2 - 48) b = 0 \\ 27 b^4 + 4 a^3 b^2 - 16 a^4 - 512 a^2 - 4096 = 0 \end{array}\right\}\right] \quad \textbf{(3.1.8)}$$

Consider now an example which has finitely many complex solutions.

```
> R := PolynomialRing([x, y, z]):
```

Define a set of equations.

```
> sys := [x^3 + y + z -1=0, x + y^3 + z -1=0, x + y + z^3 -1=0];
```

$$sys := [x^3 + y + z - 1 = 0, y^3 + x + z - 1 = 0, z^3 + x + y - 1 = 0] \quad \textbf{(3.1.9)}$$

Compute the real solutions of **sys**.

```
> dec := RealTriangularize(sys, R);
```

$$dec := [regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system] \quad \textbf{(3.1.10)}$$

```
> Display(dec, R);
```

$$\left[\left\{\begin{array}{l} x - z = 0 \\ y - z = 0 \\ z^3 + 2z - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x - 1 = 0 \\ y + 1 = 0 \\ z - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x = 0 \\ y = 0 \\ z - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x + 1 = 0 \\ y - 1 = 0 \\ z - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x - 1 = 0 \\ y = 0 \\ z = 0 \end{array}\right., \left\{\begin{array}{l} x = 0 \\ y - 1 = 0 \\ z = 0 \end{array}\right., \left\{\begin{array}{l} x - 1 = 0 \\ y - 1 = 0 \\ z + 1 = 0 \end{array}\right.\right] \quad \textbf{(3.1.11)}$$

Returning now to systems which have infinitely many complex solutions, consider now the intersection of the algebraic surfaces Sofa and Cylinder from the Algebraic Surface Gallery. As their names suggest, they are respectively the equations of a sofa and a (sort of) cylinder. One expects to find a real curve in their intersection, as we shall verify.

```
> R := PolynomialRing([z, y, x]);
  Sofa := x^2 + y^3 + z^5;
  Cyl := x^4 + z^2 - 1;
```

$$R := polynomial\_ring$$
$$Sofa := z^5 + y^3 + x^2$$
$$Cyl := x^4 + z^2 - 1 \quad \textbf{(3.1.12)}$$

```
> RealTriangularize([Sofa, Cyl], R, output=record);
```

(3.1.13)

$$\left[\left\{\begin{array}{c}(x^8-2x^4+1)z+y^3+x^2=0\\ y^6+2x^2y^3+x^{20}-5x^{16}+10x^{12}-10x^8+6x^4-1=0\\ x<1\\ x+1>0\\ x^{12}-4x^8+5x^4-1\neq 0\end{array}\right.,\quad \left\{\begin{array}{c}(x^8-2x^4+1)z-x^2=0\\ y^3+2x^2=0\\ x^{12}-4x^8+5x^4-1=0\end{array}\right.,\quad \left\{\begin{array}{c}(x^8-2x^4+1)z+x^2=0\\ y=0\\ x^{12}-4x^8+5x^4-1=0\end{array}\right.,\quad \left\{\begin{array}{c}z=0\\ y+1=0\\ x+1=0\end{array}\right.,\quad \left\{\begin{array}{c}z=0\\ y+1=0\\ x-1=0\end{array}\right.\right] \tag{3.1.13}$$

3.2 LazyRealTriangularize: interactive solving for systems of equations, inequations and inequalities

Consider again the generic equation of degree two. Now we solve it interactively.

```
> R := PolynomialRing([x, c, b, a]);
  sys := [a*x^2+b*x+c=0];
```

$$R := polynomial\_ring$$
$$sys := [x^2 a + x b + c = 0] \tag{3.2.1}$$

Use **LazyRealTriangularize** to start the decomposition.

```
> dec := LazyRealTriangularize(sys, R);
```

$$dec := \left\{\begin{array}{c}[[a x^2+b x+c=0]]\\ LazyRealTriangularize([a=0, a x^2+b x+c=0], polynomial\_ring)\\ LazyRealTriangularize([-4 c a+b^2=0, a x^2+b x+c=0], polynomial\_ring)\\ [\,]\end{array}\right.$$

Go one step further, computing components in lower dimension.

```
> dec2 := value(dec);
```

$$dec2 := \begin{cases} [[x^2 a + x b + c = 0]] & \\ \begin{cases} [[x b + c = 0, a = 0]] & \\ LazyRealTriangularize([a = 0, b = 0, x b + c = 0, x^2 a + x b + c = 0], & \end{cases} & \\ \begin{cases} [[2 a x + b = 0, 4 a c - b^2 = 0]] & \\ LazyRealTriangularize([a = 0, -4 a c + b^2 = 0, 4 a c - b^2 = 0, 2 a x + b = 0, x^2 a + & \end{cases} & \\ [\,] & \end{cases}$$

Go the last step, computing components in dimension zero.

```
> value(dec2);
```

$$\begin{cases} [[x^2 a + x b + c = 0]] & 0 < -4 a c + b^2 \wedge a \neq 0 \\ \begin{cases} [[x b + c = 0, a = 0]] & b \neq 0 \\ [[c = 0, b = 0, a = 0]] & b = 0 \end{cases} & a = 0 \\ \begin{cases} [[2 a x + b = 0, 4 a c - b^2 = 0]] & a \neq 0 \\ [[c = 0, b = 0, a = 0]] & a = 0 \end{cases} & -4 a c + b^2 = 0 \\ [\,] & otherwise \end{cases} \qquad \textbf{(3.2.4)}$$

If one is only interested in computing the main components, the list output format does the job.

```
> dec := LazyRealTriangularize(sys, R, output=list);
```

$$dec := [regular\_semi\_algebraic\_system] \qquad \textbf{(3.2.5)}$$

```
> Display(dec, R);
```

$$\left[\left\{\begin{array}{c} a x^2+b x+c=0 \\ -4 c a+b^2>0 \text{ and } a \neq 0 \end{array}\right.\right] \tag{3.2.6}$$

Another way to conduct the computation interactively is to use the record output format.

```
> dec := [LazyRealTriangularize(sys, R, output=record)];
```

$$dec := \left[\left\{\begin{array}{c} a x^2+b x+c=0 \\ -4 c a+b^2>0 \\ a \neq 0 \end{array}\right., LazyRealTriangularize([a=0, a x^2+b x+c=0], polynomial\_ring, output=record), LazyRealTriangularize([-4 c a+b^2=0, a x^2+b x+c=0], polynomial\_ring, output=record)\right] \tag{3.2.7}$$

Go one step further.

```
> dec2 := value(dec);
```

$$dec2 := \left[\left\{\begin{array}{c} a x^2+b x+c=0 \\ -4 c a+b^2>0 \\ a \neq 0 \end{array}\right., \left\{\begin{array}{c} b x+c=0 \\ b \neq 0 \\ a=0 \end{array}\right., LazyRealTriangularize([a=0, b=0, b x+c=0, a x^2+b x+c=0], polynomial\_ring, output=record), \left\{\begin{array}{c} 2 a x+b=0 \\ 4 a c-b^2=0 \\ a \neq 0 \end{array}\right., LazyRealTriangularize([a=0, -4 a c+b^2=0, 4 a c-b^2=0, 2 a x+b=0, a x^2+b x+c=0], polynomial\_ring, output=record)\right] \tag{3.2.8}$$

Go the last step.

```
> value(dec2);
```

$$\left[\left\{\begin{array}{c} a x^2+b x+c=0 \\ -4 a c+b^2>0 \\ a \neq 0 \end{array}\right., \left\{\begin{array}{c} b x+c=0 \\ b \neq 0 \\ a=0 \end{array}\right., \left\{\begin{array}{c} c=0 \\ b=0 \\ a=0 \end{array}\right., \right. \tag{3.2.9}$$

$$\left[\left\{\begin{array}{c} 2\,a\,x+b=0 \\ 4\,a\,c-b^2=0 \\ a\neq 0 \end{array}\right.,\left\{\begin{array}{c} c=0 \\ b=0 \\ a=0 \end{array}\right.\right]$$

Computing a lazy triangular decomposition is usually much less expensive than computing a full one. The following is an example.

```
> unassign('u'); variables := [x, u, v, w];
  R := PolynomialRing(variables);
  sys := [u*x^2+v*x+1=0, v*x^3+w*x+u=0, w*x^2+v*x+u<=0];
```

$$variables := [x, u, v, w]$$

$$R := polynomial\_ring$$

$$sys := \left[u\,x^2+v\,x+1=0,\ v\,x^3+w\,x+u=0,\ w\,x^2+v\,x+u\leq 0\right] \qquad \textbf{(3.2.10)}$$

Computing a lazy decomposition takes than a second.

```
> dec := LazyRealTriangularize(sys,R,output=list);
```

$$dec := [regular\_semi\_algebraic\_system] \qquad \textbf{(3.2.11)}$$

Computing a full one does not terminate within an hour.

3.3 SamplePoints: computing at least one point per connected component

Consider again the generic equation of degree two.

```
> R := PolynomialRing([x, c, b, a]);
  F := [a*x^2+b*x+c=0];
```

$$R := polynomial\_ring$$

$$F := \left[a\,x^2+b\,x+c=0\right] \qquad \textbf{(3.3.1)}$$

Compute sample points of the 4-variable hypersurface it defines.

```
> S := SamplePoints(F, R, output=record);
```

$$S := \left[\left\{\begin{array}{c} x=-1 \\ c=\frac{1}{2} \\ b=0 \\ a=-\frac{1}{2} \end{array}\right.,\left\{\begin{array}{c} x=1 \\ c=\frac{1}{2} \\ b=0 \\ a=-\frac{1}{2} \end{array}\right.,\left\{\begin{array}{c} x=-1 \\ c=-\frac{1}{2} \\ b=0 \\ a=\frac{1}{2} \end{array}\right.,\left\{\begin{array}{c} x=1 \\ c=-\frac{1}{2} \\ b=0 \\ a=\frac{1}{2} \end{array}\right.,\right] \qquad \textbf{(3.3.2)}$$

$$\left\{ x = 0,\ c = 0,\ b = -\frac{1}{2},\ a = 0 \right\},\ \left\{ x = 0,\ c = 0,\ b = \frac{1}{2},\ a = 0 \right\},\ \left\{ x = 0,\ c = 0,\ b = 0,\ a = 0 \right\},\ \left\{ x = 0,\ c = 0,\ b = 0,\ a = -\frac{1}{2} \right\},\ \left\{ x = 0,\ c = 0,\ b = 0,\ a = \frac{1}{2} \right\}$$

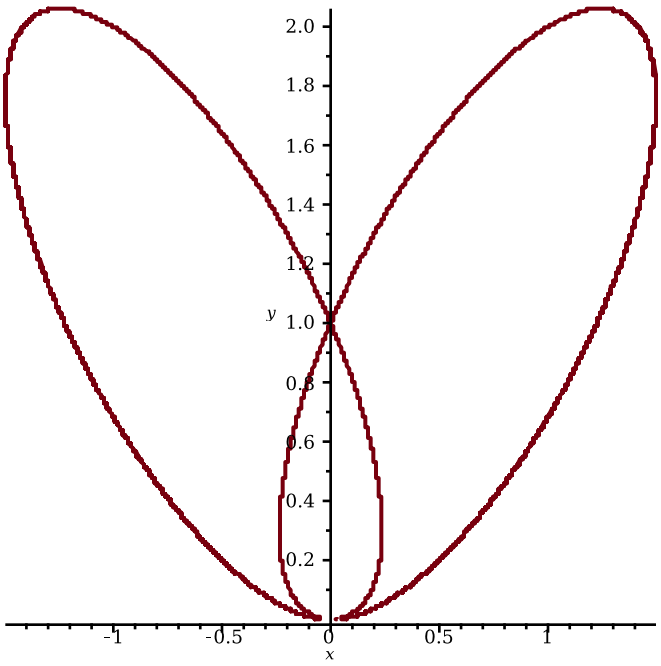
Consider the Tacnode curve. We look for sample points in the middle of the right branch.

```
> R := PolynomialRing([y, x]);
  F := [y^4-2*y^3+y^2-3*x^2*y+2*x^4];
```

$$R := polynomial\_ring$$

$$F := [2x^4 + y^4 - 3x^2y - 2y^3 + y^2] \qquad \textbf{(3.3.3)}$$

```
> with(plots):
  implicitplot(F, x=-2..2, y=-1..3, numpoints=1000000);
```




```
> SamplePoints([op(F), 2*x > 1, x < 1], R, output=record);
```

$$\left\{ y = \left[\frac{13}{32}, \frac{105}{256} \right],\ x = \frac{3}{4} \right\},\ \left\{ y = \left[\frac{237}{128}, \frac{119}{64} \right],\ x = \frac{3}{4} \right\} \qquad \textbf{(3.3.4)}$$

3.4 Isolating the real roots of a regular chain

Isolating real roots

Define a simple regular chain.

```
> R := PolynomialRing([y, x]):
  rc := Chain([2*x^2-1, 3*y^3-x], Empty(R), R):
```

Isolate its real roots.

```
> rr := RealRootIsolate(rc, R);
```

$$rr := [box, box] \tag{3.4.1.1}$$

Display the box values.

```
> solution := map(BoxValues, rr, R);
```

$$solution := \left[\left[y = \left[-\frac{20727073}{33554432}, -\frac{20727063}{33554432}\right], x = \left[-\frac{46341}{65536}, -\frac{741455}{1048576}\right]\right], \left[y = \left[\frac{10363531}{16777216}, \frac{10363537}{16777216}\right], x = \left[\frac{741455}{1048576}, \frac{46341}{65536}\right]\right]\right] \tag{3.4.1.2}$$

Each element of the list is called a box. Each box encodes a real root, in the sense that it contains exactly one real root of rc. The RealRootIsolate command returns all real roots of a given regular chain in this way. Thus, it is guaranteed that no root is lost. In the example, there are two real roots.

```
> solution[1]; solution[2];
```

$$\left[y = \left[-\frac{20727073}{33554432}, -\frac{20727063}{33554432}\right], x = \left[-\frac{46341}{65536}, -\frac{741455}{1048576}\right]\right]$$
$$\left[y = \left[\frac{10363531}{16777216}, \frac{10363537}{16777216}\right], x = \left[\frac{741455}{1048576}, \frac{46341}{65536}\right]\right] \tag{3.4.1.3}$$

Thus, the first root satisfies $\frac{1}{2} < x < 1$ and $0 < y < 1$, and the second one satisfies $-1 < x < -\frac{1}{2}$ and $-1 < x < 0$.

The BoxValues command returns a list of the form $v = i$, where v is a variable and i is either a rational number or an open interval encoded by a list. The reason why singletons are returned is that rational solutions are sometimes hit, as in the following example.

```
> rc := Chain([2*x-1, 3*y^3-x], Empty(R), R):
  rr := RealRootIsolate(rc, R):
  map(BoxValues, rr, R);
```

$$\left[\left[y = \left[\frac{577053}{1048576}, \frac{288527}{524288}\right], x = \frac{1}{2}\right]\right] \tag{3.4.1.4}$$

The Display command prints the isolated real roots in a pretty manner.

```
> Display(rr, R);
```

(3.4.1.5)

$$\left[\left\{\begin{array}{c} y=\left[\frac{577053}{1048576},\frac{288527}{524288}\right] \\ x=\frac{1}{2} \end{array}\right\}\right] \qquad \textbf{(3.4.1.5)}$$

Choosing a precision

The isolating boxes can be as small as needed. The *abserr* option allows you to obtain boxes whose widths are smaller than or equal to a certain absolute precision.

```
> rc := Chain([2*x^2-1, 3*y^3-x], Empty(R), R):
  rr := RealRootIsolate(rc, R, abserr = 1/10^10):
  map(BoxValues, rr, R);
```

$$\left[\left[y=\left[-\frac{2716737851577}{4398046511104},-\frac{10866951406269}{17592186044416}\right],x=\left[-\frac{759250125}{1073741824},-\frac{97184015999}{137438953472}\right]\right],\left[y=\left[\frac{2716737851567}{4398046511104},\frac{2716737851577}{4398046511104}\right],x=\left[\frac{97184015999}{137438953472},\frac{759250125}{1073741824}\right]\right]\right] \qquad \textbf{(3.4.2.1)}$$

```
> evalf(%);
```

$$[[y=[-0.6177146705,-0.6177146705],x=[-0.7071067812,-0.7071067812]],[y=[0.6177146705,0.6177146705],x=[0.7071067812,0.7071067812]]] \qquad \textbf{(3.4.2.2)}$$

In fact, this involves the algebraic numbers $\sqrt{\frac{1}{2}}$ and $\left(\frac{\sqrt{\frac{1}{2}}}{3}\right)^{\frac{1}{3}}$.

```
> evalf(sqrt(1/2)); evalf(((sqrt(1/2))/3)^(1/3));
```

$$0.7071067810$$
$$0.6177146705 \qquad \textbf{(3.4.2.3)}$$

3.5 Isolating and counting the real zeros of a semi-algebraic system with finitely many solutions

Recall that a semi-algebraic system (SAS) is a system containing polynomial equations, polynomial (non-strict or/and strict) inequalities, and polynomial inequations, which are denoted by F, N, P, and H, respectively. For example, the system $\{x^2+y^2-1=0, xy-1=0, x\geq 0, y>0\}$ is represented by

```
> F := [x^2+y^2-1, 2*x*y-1]:
  N := [x]:
  P := [y]:
  H := []:
```

Define the polynomial ring

```
> R := PolynomialRing([x, y]):
```

The RealRootIsolate command isolates all the real zeros of a given semi-algebraic system.

```
> rr := RealRootIsolate(F, N, P, H, R);
  solution := Display(rr, R);
```

$$rr := [box]$$

$$solution := \left[\left\{ x = \left[\frac{5}{8}, \frac{3}{4}\right],\ y = \left[\frac{45}{64}, \frac{91}{128}\right] \right\}\right] \qquad \textbf{(3.5.1)}$$

The RealRootCounting command computes the number of distinct real solutions of a given semi-algebraic system.

```
> RealRootCounting(F, N, P, H, R);
```

$$1 \qquad \textbf{(3.5.2)}$$

Consider a less trivial example.

```
> R := PolynomialRing([z, y, x, c]):
  F := [1-c*x-x*y^2-x*z^2, 1-c*y-y*x^2-y*z^2, 1-c*z-z*x^2-z*y^2, 8*
  c^6+378*c^3-27]:
  N := []:
  P := [c, 1-c]:
  H := []:
  RealRootCounting(F, N, P, H, R);
```

$$4 \qquad \textbf{(3.5.3)}$$

If the input system has infinitely many complex solutions (that is, it is positive-dimensional over the complex numbers), RealRootCounting will return a message suggesting to call RealRootClassification instead.

```
> R := PolynomialRing([x, y]):
  RealRootCounting([x^2+y^2], [], [], [], R);
Error, (in RegularChains:-SemiAlgebraicSetTools:-RealRootCounting)
system is not zero-dimensional; try RealRootClassification
```

3.6 Partial cylindrical algebraic decomposition

The command RealRootClassification makes use of Partial Cylindrical Algebraic Decomposition (PCAD). The calling sequence is PartialCylindricalAlgebraicDecomposition(p, lp, R), where R is a polynomial ring, p is a polynomial in R, and lp is a list of polynomials in R representing positivity conditions. The output of the function is a list *sp* of *sample points*. Each sample point is represented by a list of rational numbers, as many as there are variables in R. Each inner list gives the coordinates of a sample point of a (real) open connected component of the (real) space decomposed by the equation $p = 0$. Moreover, sample points which do not satisfy $q > 0$ for all polynomials $q \in lp$ are discarded.

```
> R := PolynomialRing([x, y, t]):
  PartialCylindricalAlgebraicDecomposition(y, [], R);
  PartialCylindricalAlgebraicDecomposition(y, [x], R);
  PartialCylindricalAlgebraicDecomposition(x*y, [], R);
  PartialCylindricalAlgebraicDecomposition(x*y, [x], R);
```

$$\left[\left[0, -\frac{1}{2}, 0\right], \left[0, \frac{1}{2}, 0\right]\right]$$

$$\left[\left[\frac{1}{2}, -\frac{1}{2}, 0\right], \left[\frac{1}{2}, \frac{1}{2}, 0\right]\right]$$

$$\left[\left[-\frac{1}{2}, -\frac{1}{2}, 0\right], \left[\frac{1}{2}, -\frac{1}{2}, 0\right], \left[-\frac{1}{2}, \frac{1}{2}, 0\right], \left[\frac{1}{2}, \frac{1}{2}, 0\right]\right]$$

$$\left[\left[\frac{1}{2}, -\frac{1}{2}, 0\right], \left[\frac{1}{2}, \frac{1}{2}, 0\right]\right] \qquad \textbf{(3.6.1)}$$

As a more advanced example, compute sample points for the polynomial system of equations *sys* below which in addition make each polynomial in *sys* positive.

```
> R := PolynomialRing([x, y]):
  sys := [-x^2-y+1, x+y^2-1, -x^2+y^2]:
  sp := PartialCylindricalAlgebraicDecomposition(product(sys[j], j = 1 .. nops(sys)), sys, R)
```

$$sp := \left[\left[0, -\frac{17}{8}\right], \left[\frac{221}{512}, -\frac{287}{256}\right], \left[\frac{197}{512}, \frac{13}{16}\right]\right] \qquad \textbf{(3.6.2)}$$

Check the result.

```
> for i to nops(sp) do
      print(eval(sys, [x = sp[i][1], y = sp[i][2]]))
  end do;
```

$$\left[\frac{25}{8}, \frac{225}{64}, \frac{289}{64}\right]$$

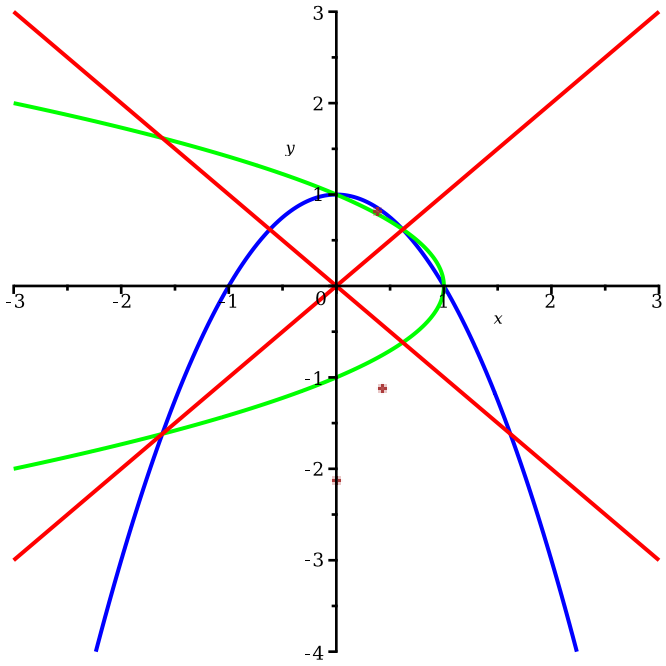
$$\left[\frac{507191}{262144}, \frac{45121}{65536}, \frac{280635}{262144}\right]$$

$$\left[\frac{10343}{262144}, \frac{23}{512}, \frac{134247}{262144}\right] \qquad \textbf{(3.6.3)}$$

Plot these sample points.

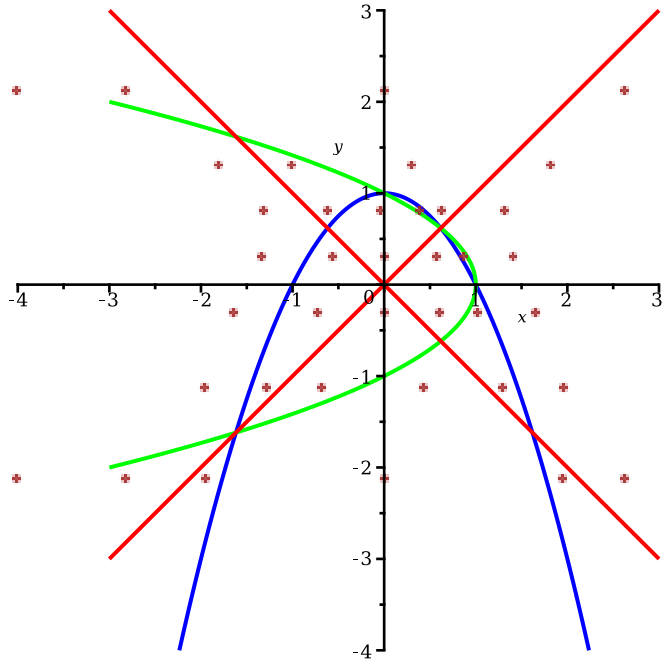
```
> with(plots):
  colors := [blue, green, red, brown]:
```

```
curves := seq(implicitplot(sys[i], x = -3 .. 3, y = -4 .. 3,
color = colors[i], numpoints = 5000), i = 1 .. 3):
points := pointplot(sp, color = colors[4]):
display([curves, points]);
```



Compute and plot sample points for the same system *sys* but without any sign constraints.

```
> sp := PartialCylindricalAlgebraicDecomposition(product(sys[j], j = 1 .. nops(sys)), [], R):
points := pointplot(sp, color = colors[4]):
display([curves, points]);
```



3.7 Cylindrical algebraic decomposition

Cylindrical Algebraic Decomposition (CAD) is a fundamental and powerful tool for studying systems of equations, inequations and inequalities.

Our algorithm is different from the traditional algorithm of Collins. It first computes a cylindrical decomposition of the complex space, from which a CAD of the real space can be easily extracted.

Consider the hyperbola $xy-1=0$.

```
> R := PolynomialRing([y, x]);
  F := [y*x-1];
```

$$R := polynomial\_ring$$
$$F := [xy-1] \qquad \textbf{(3.7.1)}$$

A cylindrical algebraic decomposition adapted to the polynomial $xy-1$ can be computed by the command CylindricalAlgebraicDecompose as follows:

```
> outcad := CylindricalAlgebraicDecompose(F, R, output=piecewise);
```

$$outcad := \begin{cases} \begin{cases} 1 & y < \frac{1}{x} \\ 1 & y = \frac{1}{x} \\ 1 & \frac{1}{x} < y \end{cases} & x < 0 \\ 1 & x = 0 \\ \begin{cases} 1 & y < \frac{1}{x} \\ 1 & y = \frac{1}{x} \\ 1 & \frac{1}{x} < y \end{cases} & 0 < x \end{cases} \qquad \textbf{(3.7.2)}$$

The output CAD is described by a nested piecewise function. The outmost piecewise function is a function with three conditions $x<0$, $x=0$, and $0<x$. Each of the conditions has a corresponding expression, which is again a piecewise function. The output could be read from top to bottom and from right to left.

One can see that the CAD consists of seven cells.

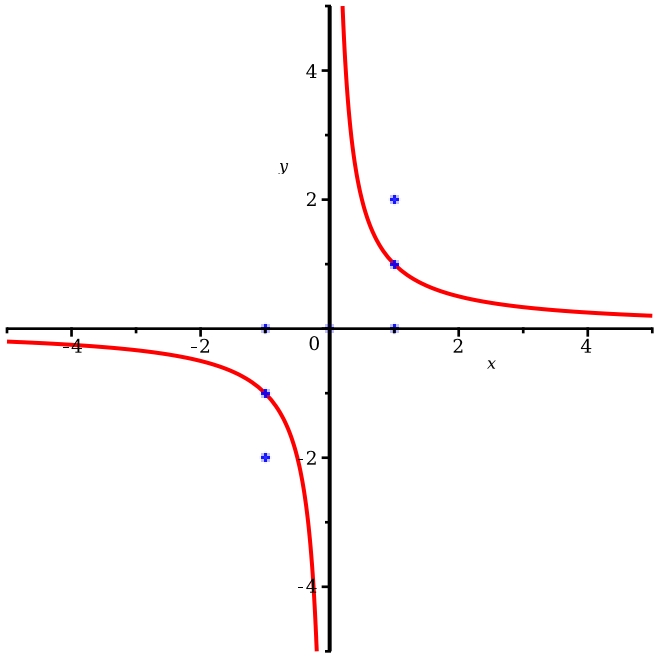
For example,

$x < 0 \,\mathbf{and}\, y < \frac{1}{x}$ describes one cell of the CAD, where

$[regular\_chain, [[-1,-1],[-2,-2]]]$
represents a sample point in this cell.
This sample point is represented by a regular chain and an isolating box such that inside this box there is one and only one root of this regular chain.

We plot the hyperbola and all the sample points of the CAD adapted to this hyperloa as follows.

```
> with(plots):
  sp := [[-1, -2], [-1, -1], [-1, 0], [0, 0], [1, 0], [1, 1], [1,
  2]]:
  points := pointplot(sp, color = blue):
  curve := implicitplot([x*y-1, x], x = -5 .. 5, y = -5 .. 5,
  color=[red, black]):
  display([curve, points]);
```



The piecewise format is good when the output has few cells. When many cells are present the **'output'='cadcell'** and **'output'='rootof'** formats are useful.

```
> R := PolynomialRing([x, c, b, a]);
  F := [a*x^2+b*x+c];
  cad := CylindricalAlgebraicDecompose(F, R, output = cadcell);
```

$$R := polynomial\_ring$$
$$F := [x^2 a + x b + c]$$
$$cad := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, \quad \textbf{(3.7.3)}$$

```
    cad_cell, cad_cell, cad_cell, cad_cell, cad_cell, cad_cell, cad_cell,
    cad_cell, cad_cell, cad_cell, cad_cell, cad_cell, cad_cell, cad_cell,
    cad_cell, cad_cell, cad_cell, cad_cell, cad_cell, cad_cell]
```

```
> nops(cad);
```

$$27 \qquad \textbf{(3.7.4)}$$

The output consists of 27 cells, which can also been displayed as so-called Taski Formulas.

```
> Display(cad, R);
```

$$\left[\left\{\begin{array}{l} x = x \\ c < \frac{b^2}{4\,a} \\ b = b \\ a < 0 \end{array}\right\}, \left\{\begin{array}{l} x < -\frac{b}{2\,a} \\ c = \frac{b^2}{4\,a} \\ b = b \\ a < 0 \end{array}\right\}, \left\{\begin{array}{l} x = -\frac{b}{2\,a} \\ c = \frac{b^2}{4\,a} \\ b = b \\ a < 0 \end{array}\right\}, \left\{\begin{array}{l} -\frac{b}{2\,a} < x \\ c = \frac{b^2}{4\,a} \\ b = b \\ a < 0 \end{array}\right\}, \qquad \textbf{(3.7.5)}$$

$$\left\{\begin{array}{l} x < \frac{-b+\sqrt{-4\,c\,a+b^2}}{2\,a} \\ \frac{b^2}{4\,a} < c \\ b = b \\ a < 0 \end{array}\right\}, \left\{\begin{array}{l} x = \frac{-b+\sqrt{-4\,c\,a+b^2}}{2\,a} \\ \frac{b^2}{4\,a} < c \\ b = b \\ a < 0 \end{array}\right\},$$

$$\left\{\begin{array}{l} \frac{-b+\sqrt{-4\,c\,a+b^2}}{2\,a} < x \wedge x < -\frac{b+\sqrt{-4\,c\,a+b^2}}{2\,a} \\ \frac{b^2}{4\,a} < c \\ b = b \\ a < 0 \end{array}\right\},$$

$$\left\{\begin{array}{l} x = -\frac{b+\sqrt{-4\,c\,a+b^2}}{2\,a} \\ \frac{b^2}{4\,a} < c \\ b = b \\ a < 0 \end{array}\right\}, \left\{\begin{array}{l} -\frac{b+\sqrt{-4\,c\,a+b^2}}{2\,a} < x \\ \frac{b^2}{4\,a} < c \\ b = b \\ a < 0 \end{array}\right\},$$

$$\left\{ x < -\frac{c}{b} \;\middle|\; c = c \wedge b < 0 \wedge a = 0 \right\},\ \left\{ x = -\frac{c}{b} \;\middle|\; c = c \wedge b < 0 \wedge a = 0 \right\},\ \left\{ -\frac{c}{b} < x \;\middle|\; c = c \wedge b < 0 \wedge a = 0 \right\},\ \left\{ x = x \;\middle|\; c < 0 \wedge b = 0 \wedge a = 0 \right\},$$

$$\left\{ x = x \;\middle|\; c = 0 \wedge b = 0 \wedge a = 0 \right\},\ \left\{ x = x \;\middle|\; 0 < c \wedge b = 0 \wedge a = 0 \right\},\ \left\{ x < -\frac{c}{b} \;\middle|\; c = c \wedge 0 < b \wedge a = 0 \right\},\ \left\{ x = -\frac{c}{b} \;\middle|\; c = c \wedge 0 < b \wedge a = 0 \right\},$$

$$\left\{ -\frac{c}{b} < x \;\middle|\; c = c \wedge 0 < b \wedge a = 0 \right\},\ \left\{ x < -\frac{b + \sqrt{-4\,c\,a + b^2}}{2\,a} \;\middle|\; c < \frac{b^2}{4\,a} \wedge b = b \wedge 0 < a \right\},$$

$$\left\{ x = -\frac{b + \sqrt{-4\,c\,a + b^2}}{2\,a} \;\middle|\; c < \frac{b^2}{4\,a} \wedge b = b \wedge 0 < a \right\},$$

$$\left\{ -\frac{b + \sqrt{-4\,c\,a + b^2}}{2\,a} < x \wedge x < \frac{-b + \sqrt{-4\,c\,a + b^2}}{2\,a} \;\middle|\; c < \frac{b^2}{4\,a} \wedge b = b \wedge 0 < a \right\},$$

$$\left[\left\{x = \frac{-b+\sqrt{-4\,c\,a+b^2}}{2\,a},\ c < \frac{b^2}{4\,a},\ b = b,\ 0 < a\right\},\ \left\{\frac{-b+\sqrt{-4\,c\,a+b^2}}{2\,a} < x,\ c < \frac{b^2}{4\,a},\ b = b,\ 0 < a\right\},\right.$$

$$\left.\left\{x < -\frac{b}{2\,a},\ c = \frac{b^2}{4\,a},\ b = b,\ 0 < a\right\},\ \left\{x = -\frac{b}{2\,a},\ c = \frac{b^2}{4\,a},\ b = b,\ 0 < a\right\},\ \left\{-\frac{b}{2\,a} < x,\ c = \frac{b^2}{4\,a},\ b = b,\ 0 < a\right\},\ \left\{x = x,\ \frac{b^2}{4\,a} < c,\ b = b,\ 0 < a\right\}\right]$$

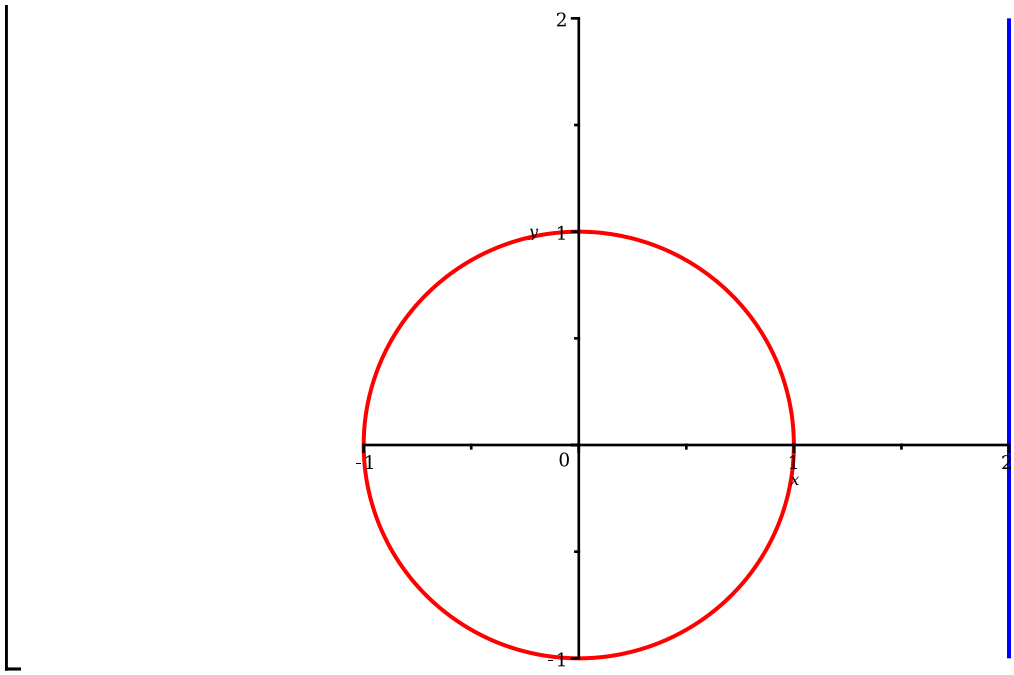
Another important feature of the CAD command is the fact that it can take a list of semi-algebraic systems as input. This list of semi-algebraic systems represents a DNF quantifier free formula, that is, a disjunction of semi-algebraic systems.

Consider the union of a cirle and a line.

```
> Circle := x^2+y^2-1=0;
  Line := x-2=0;
  with(plots):
  implicitplot([Circle, Line], x = -1 .. 3, y = -1 .. 2, color=
  [red, blue]);
```

$$Circle := x^2 + y^2 - 1 = 0$$

$$Line := x - 2 = 0$$



Compute a CAD of the union of *Circle* and *Line*.

```
> R := PolynomialRing([y, x]);
  cad := CylindricalAlgebraicDecompose([[Circle], [Line]], R,
  output=cadcell);
```

$$R := polynomial\_ring$$

$$cad := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell] \quad \textbf{(3.7.6)}$$

```
> Display(cad, R);
```

$$\left[\left\{\begin{array}{l} y = 0 \\ x = -1 \end{array}\right\}, \left\{\begin{array}{l} y = -\sqrt{-x^2+1} \\ -1 < x \wedge x < 1 \end{array}\right\}, \left\{\begin{array}{l} y = \sqrt{-x^2+1} \\ -1 < x \wedge x < 1 \end{array}\right\}, \left\{\begin{array}{l} y = 0 \\ x = 1 \end{array}\right\}, \left\{\begin{array}{l} y = y \\ x = 2 \end{array}\right\}\right] \quad \textbf{(3.7.7)}$$

Each cad_cell has a sample point associated with it.

```
> sp := map(SamplePoints, cad, R): Display(sp,R);
```

$$\left[\left\{\begin{array}{l} y = 0 \\ x = -1 \end{array}\right\}, \left\{\begin{array}{l} y = -1 \\ x = 0 \end{array}\right\}, \left\{\begin{array}{l} y = 1 \\ x = 0 \end{array}\right\}, \left\{\begin{array}{l} y = 0 \\ x = 1 \end{array}\right\}, \left\{\begin{array}{l} y = 0 \\ x = 2 \end{array}\right\}\right] \quad \textbf{(3.7.8)}$$

Last but not least, the CAD command is back engine of SoveTools:-SemiAlgebraic.

```
> SolveTools:-SemiAlgebraic([x^2+x-c>0], [c, x]);
```

$$\left[\left[c < -\frac{1}{4}, x = x\right], \left[c = -\frac{1}{4}, x \neq -\frac{1}{2}\right], \left[-\frac{1}{4} < c, x < -\frac{1}{2} - \frac{\sqrt{4c+1}}{2}\right], \left[-\frac{1}{4} < c, -\frac{1}{2} + \frac{\sqrt{4c+1}}{2} < x\right]\right] \quad (3.7.9)$$

```
> cad := CylindricalAlgebraicDecompose([[x^2+x-c>0]],
  PolynomialRing([x, c]), output=rootof);
```

$$cad := \left[\left[c < -\frac{1}{4}, x = x\right], \left[c = -\frac{1}{4}, x \neq -\frac{1}{2}\right], \left[-\frac{1}{4} < c, x < -\frac{1}{2} - \frac{\sqrt{4c+1}}{2}\right], \left[-\frac{1}{4} < c, -\frac{1}{2} + \frac{\sqrt{4c+1}}{2} < x\right]\right] \quad (3.7.10)$$

3.8 Solving a linear semi-algebraic system

The input linear semi-algebraic system:

```
> S := [f00 = c, f10 = c + cx1, f01 = c + cx2,
        f11 = c + cx1 + cx2 + cx1x2,
        cx1x2 <= 0];
```

$$S := [f00 = c, f10 = c + cx1, f01 = c + cx2, f11 = c + cx1 + cx2 + cx1x2, cx1x2 \leq 0] \quad (3.8.1)$$

Define the elimination order (descending order).

```
> R := PolynomialRing([cx1x2, cx1, cx2, c, f00, f01, f10, f11]):
```

Call LinearSolve to eliminte the variables.

The output is a set of equivalent linear equationsand inequalities sorted in ascending order according to the larget variables appearing in the constraints. It provides conditions on lower order variables such that higher order variables having solutions.

In other words, the projection of the solutions of input system S onto any lower dimensional space, say the space formed by the smallest i variables, are exactly the solutions of those constraints in the output which only involve the smallest i smallest variables.

```
> R := PolynomialRing([cx1x2, cx1, cx2, c, f00, f01, f10, f11]):

  result := LinearSolve(S, R);
```

$$result := [f00 \le -f11 + f10 + f01, c = f00, cx2 = f01 - f00, cx1 = f10 - f00, cx1x2 = f11 - f10 - f01 + f00] \quad \textbf{(3.8.2)}$$

3.9 Verifying the output of different real solvers

On a given input polynomial system, two solving tools may produce correct results that look fairly different. Proving that these two results, say S1 and S2, are equivalent can be a very complex task. Here's an example.

Given a triangle with edges a, b, c (denoting the respective lengths by a, b, c as well) the following two conditions $C1, C2$ are both characterizing the fact that the external bisector of the angle of a, c intersects with b on the other side of a than the triangle:

We set S1 and S2 up first. S1 is the disjuction of C1 and C2.

```
> C1:=[a > 0 , b > 0 , c > 0 , a < b + c ,
  b < a + c , c < a + b,  b^2 + a^2 - c^2 <= 0 ];

  C2:=[a > 0 , b > 0 , c > 0 , a < b + c ,
  b < a + c , c < a + b,
  c*(b^2 + a^2 - c^2)^2 < a*b^2*(2*a*c - (c^2 + a^2 - b^2))]:


  S1:=[C1, C2];

  S2 := [a-c<0,  a > 0 , b > 0 ,
  c > 0 , a < b + c , b < a + c , c < a + b];
```

$$C1 := [0 < a, 0 < b, 0 < c, a < b + c, b < a + c, c < a + b, a^2 + b^2 - c^2 \le 0]$$

$$S1 := [[0 < a, 0 < b, 0 < c, a < b + c, b < a + c, c < a + b, a^2 + b^2 - c^2 \le 0], [0 < a, 0 < b, 0 < c, a < b + c, b < a + c, c < a + b, c\,(a^2 + b^2 - c^2)^2 < a\,b^2\,(-a^2 + 2\,c\,a + b^2 - c^2)]]$$

$$S2 := [a - c < 0, 0 < a, 0 < b, 0 < c, a < b + c, b < a + c, c < a + b] \quad \textbf{(3.9.1)}$$

Compute regular semi-algebraic system representations dec1 (resp. dec2) for S1 (resp. S2).

```
> R := PolynomialRing([a,b,c]):
  dec1 := map(op,  map(RealTriangularize, S1,R));
```

```
dec2:= RealTriangularize(S2, R);
```

$$dec1 := [regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system]$$

$$dec2 := [regular\_semi\_algebraic\_system] \qquad \textbf{(3.9.2)}$$

Compute the differences: S1 \ S2 and S2 \ S1.

```
> Difference(dec1,dec2,R);
```

$$[\,] \qquad \textbf{(3.9.3)}$$

```
> Difference(dec2,dec1,R);
```

$$[\,] \qquad \textbf{(3.9.4)}$$

3.10 Computing the projection of a semi-algebraic set

In the following problem, we are interesting in determining sufficient and necessary conditions on the variables a1, a2, for the prescribed semi-algebraic system to have solutions in r1, r2, x1, x2, e1, e2. This question is equivalent to compute the standard projection of the corresponding semi-algebraic set on the (a1, a2)-plane.

Problem: $\exists$(r1, r2, x1, x2, e1, e2) ($\wedge_{f \in eqs}$ f=0 $\wedge_{p \in pie}$ p>0)

```
> vars := [r1, r2, x1, x2, e1, e2, a1, a2];
  eqs := [
    r1^2 - r2^2 - e1^2 + e2^2 + x1^2 - 2*x1*x2 + x2^2,
    2*r1*r2 - 2*e1*e2,
    -r1 + e1 + x1*a2 - x2*a2,
    -r2 + e2 - x1*a1 + x2*a1,
    r1^2 + r2^2 + x1^2 + x2^2 + e1^2 + e2^2 - 1];

  pie := [r1, -e1];
  ineqs := [];
  nie := [];
  params := [a1,a2];

  R := PolynomialRing(vars):
  proj := Projection(eqs, nie, pie, ineqs, 2, R);
```

$$vars := [r1, r2, x1, x2, e1, e2, a1, a2]$$

$$eqs := [-e1^2 + e2^2 + r1^2 - r2^2 + x1^2 - 2\,x1\,x2 + x2^2, -2\,e1\,e2 + 2\,r1\,r2, x1\,a2 - x2\,a2 + e1 - r1, -x1\,a1 + x2\,a1 + e2 - r2, e1^2 + e2^2 + r1^2 + r2^2 + x1^2 + x2^2 - 1]$$

$$pie := [r1, -e1]$$
$$ineqs := [\,]$$
$$nie := [\,]$$
$$params := [a1, a2]$$
$$proj := [regular\_semi\_algebraic\_system, regular\_semi\_algebraic\_system] \quad \textbf{(3.10.1)}$$

Show the result:

```
> Display(proj, R);
```

$$\left[\left\{\begin{array}{l} a1 = 0 \\ \left\{\begin{array}{l} a2 < -1 \\ \textbf{or}\ a2 - 1 > 0 \end{array}\right. \end{array}\right., a1^2 + a2^2 - 1 > 0 \ \textbf{and}\ a1 \neq 0 \ \textbf{and}\ a2 \neq 0\right] \quad \textbf{(3.10.2)}$$

4. Linear algebra over towers of (field) extensions

Regular chains encode towers of transcendental and algebraic extensions of the base field. In practice, these towers themselves are not always fields and may have zero-divisors. Many standard algorithms (for example, for solving systems of linear equations) require that the coefficient ring be a field. This difficulty is overcome by means of the so-called *D5 Principle*. This principle is at the core of the theory of regular chains and allows us to generalize many algorithms which are a priori restricted to coefficients from a field.

The MatrixTools submodule applies the D5 Principle to algorithms for solving linear systems with coefficient rings encoded by regular chains. Below, use of this submodule is demonstrated by two examples. As discussed earlier regarding polynomial gcd computations, splitting naturally occurs when computing modulo regular chains. Sometimes, split results can be recombined as illustrated below.

4.1 Automatic case discussion and recombination (I)

Assume that there are two algebraic entities, y and z; that they have the same square; and that z is a 4th root of -1. Suppose you need to perform algebraic computations with y and z.

```
> R := PolynomialRing([y, z]):
  rc := Chain([z^4+1, y^2-z^2], Empty(R), R):
  Equations(rc, R);
```

$$[y^2 - z^2, z^4 + 1] \quad \textbf{(4.1.1)}$$

For example, you may want to compute the inverse of the following matrix.

```
> A := [ 1   y+z ]
       [ 0   y-z ] :
```

Clearly, the result depends on whether y and z are equal or not. Note that from the assumptions above, neither $y=z$ nor $y=-z$ can be deduced. When $y \neq z$, the matrix A has an inverse. When they are equal, the matrix A is singular. These facts are detected automatically by the command MatrixInverse.

```
> result := MatrixInverse(A, rc, R);
```

$$result := \left[\left[\left[\left[\left[\begin{matrix} 1 & 0 \\ 0 & \frac{z^3}{2} \end{matrix}\right], regular\_chain\right]\right], \left[\left[\text{"noInv"}, \begin{matrix} 1 & y+z \\ 0 & y-z \end{matrix}\right], regular\_chain\right]\right]\right] \quad \textbf{(4.1.2)}$$

Check the first result. Note the use of the command MatrixMultiply in order to multiply two matrices modulo a regular chain.

```
> B, rc1 := op(result[1][1]);
  Equations(rc1, R);
  MatrixMultiply(B, A, rc1, R)
```

$$B, rc1 := \begin{bmatrix} 1 & 0 \\ 0 & \frac{z^3}{2} \end{bmatrix}, regular\_chain$$

$$[y+z, z^4+1]$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \textbf{(4.1.3)}$$

```
> rc2 := result[2][1][3]; Equations(rc2, R);
```

$$rc2 := regular\_chain$$

$$[y-z, z^4+1] \quad \textbf{(4.1.4)}$$

You can see that the computation (and the regular chain) was split into two branches. The first one corresponds to $y+z=0$, and in that branch the matrix is invertible. The second branch corresponds to $y=z$, and in that branch the matrix is not invertible.

Consider the matrix M below and compute its inverse modulo the regular chain rc. Here again, computations split. However, the matrix M is invertible in both cases.

```
> M := Matrix([[1, y+z], [2, y-z]]);
  result := MatrixInverse(M, rc, R);
```

$$M := \begin{bmatrix} 1 & y+z \\ 2 & y-z \end{bmatrix}$$

$$result := \left[\left[\left[\left[\left[\begin{matrix} 1 & 0 \\ -z^3 & \frac{z^3}{2} \end{matrix}\right], regular\_chain\right]\right], \left[\left[\begin{matrix} 0 & \frac{1}{2} \\ -\frac{z^3}{2} & \frac{z^3}{4} \end{matrix}\right], regular\_chain\right]\right], [\,]\right] \quad (4.1.5)$$

Double check this result.

```
> op(result[1][1]);
```

$$\left[\begin{matrix} 1 & 0 \\ -z^3 & \frac{z^3}{2} \end{matrix}\right], regular\_chain \quad (4.1.6)$$

```
> unassign('N');
  N[1] , rc1 := op(result[1][1]);
  Equations(rc1, R); N[1];
  MatrixMultiply(N[1], M, rc1, R)
```

$$N_1, rc1 := \left[\begin{matrix} 1 & 0 \\ -z^3 & \frac{z^3}{2} \end{matrix}\right], regular\_chain$$

$$[y+z, z^4+1]$$

$$\left[\begin{matrix} 1 & 0 \\ -z^3 & \frac{z^3}{2} \end{matrix}\right]$$

$$\left[\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}\right] \quad (4.1.7)$$

```
> N[2], rc2 := op(result[1][2]);
  Equations(rc2, R);
  MatrixMultiply(N[2], M, rc2, R)
```

$$N_2, rc2 := \left[\begin{matrix} 0 & \frac{1}{2} \\ -\frac{z^3}{2} & \frac{z^3}{4} \end{matrix}\right], regular\_chain$$

$$[y-z, z^4+1]$$

(4.1.8)

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \textbf{(4.1.8)}$$

Since the matrix M is invertible in both branches given by the regular chains rc_1 and rc_2, it is natural to ask whether there is a "generic" answer that would hold for both cases. The answer is *yes*. Technically speaking, one can observe that the Chinese Remainder Theorem applies since the (saturated) ideals of rc_1 and rc_2 are relatively prime. The command MatrixCombine implements this recombination.

```
> combined := MatrixCombine([rc1, rc2], R, [N1, N2])
```

$$combined := \left[\left[\left[\begin{matrix} \frac{yz^3}{2}+\frac{1}{2} & -\frac{yz^3}{4}+\frac{1}{4} \\ \frac{1}{4}yz^2-\frac{3}{4}z^3 & -\frac{1}{8}yz^2+\frac{3}{8}z^3 \end{matrix}\right], regular\_chain\right]\right] \quad \textbf{(4.1.9)}$$

Check that the above matrix times M gives the identity matrix modulo rc.

```
> MatrixMultiply(combined[1][1], M, combined[1][2], R);
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \textbf{(4.1.10)}$$

4.2 Automatic case discussion and recombination (II)

Can there be several cases in the output of the command MatrixCombine? In other words, is it possible that this command fails recombining several cases into one, even if the Chinese Remainder Theorem applies? Yes, this can happen, for algebraic reasons that shall be explained below. For starters, consider one of the previous polynomial systems.

```
> sys := {x+y+z^2-1, x+y^2+z-1, x^2+y+z-1}:
  R := PolynomialRing([x, y, z]):
  l := Triangularize(sys, R, normalized = yes);
  map(Equations, l, R);
```

$$l := [regular\_chain, regular\_chain, regular\_chain, regular\_chain]$$

$$[[x-z, y-z, z^2+2z-1], [x, y, z-1], [x, y-1, z], [x-1, y, z]] \quad \textbf{(4.2.1)}$$

Next, generate four random 2×2 matrices with polynomial entries.

```
> A := [seq(Matrix([seq([seq(randpoly([x, y, z], degree = 1), j = 1
  .. 2)], i = 1 .. 2)]), k = 1 .. 4)];
```

$$A := \left[\begin{bmatrix} 68x+69y+56z+89 & 65x+64y-33z-76 \\ 96x-8y+79z-64 & -75x-81y+98z+9 \end{bmatrix}, \right.$$

$$\left. \begin{bmatrix} 18x-94y-89z-1 & 85x-50y+80z-36 \\ -9x-73y-2z+29 & 25x+59y+31z-30 \end{bmatrix}, \right. \quad \textbf{(4.2.2)}$$

$$\left[\begin{bmatrix} 57x-60y+72z+41 & 34x+85y+61z \\ 96x+48y+85z-1 & 26x+28y-3z-49 \end{bmatrix}, \begin{bmatrix} 11x-79y-27z+58 & -85x-79y+91z-98 \\ 44x-16y+2z-46 & -69x-30y-37z+12 \end{bmatrix}\right]$$

Attempt the recombination of the four cases given by the regular chains in A, and observe that MatrixCombine produces two cases.

```
> combined := MatrixCombine(l, R, A);
```

$$combined := \left[\left[\begin{bmatrix} -88y+69 & 268y-183 \\ 49y-2 & 36y-57 \end{bmatrix}, regular\_chain\right], \left[\begin{bmatrix} -186z^2-179z+275 & 12z^2+120z-88 \\ -38z^2+91z-26 & 25z^2-8z-16 \end{bmatrix}, regular\_chain\right]\right] \quad (4.2.3)$$

Now investigate why these two cases cannot be merged into a single one.

```
> rc1 := combined[1][2]:
  Equations(rc1, R);
  rc2 := combined[2][2]:
  Equations(rc2, R)
```

$$[x+y-1, y^2-y, z]$$
$$[2x+z^2-1, 2y+z^2-1, z^3+z^2-3z+1] \quad (4.2.4)$$

The two ideals generated by rc_1 and rc_2 are obviously relatively prime (no common roots in z). Thus, the Chinese Remainder Theorem applies. But, if you try to recombine rc_1 and rc_2 into a single system, this will create a polynomial in y with a zero-divisor as a leading coefficient, as seen below. This is forbidden by the properties of a regular chain.

First, create two new regular chains with the polynomials from rc_1 and rc_2 that are univariate in the variable z.

```
> S := PolynomialRing([z]):
  rc3 := Chain([z], Empty(S), (S)):
  rc4 := Chain([z^3+z^2-3z+1], Empty(S), S):
```

Then define two 1×1 matrices U_1, U_2, and combine them with regards to rc_3 and rc_4.

```
> U_1 := Matrix([[y^2-y]]);
  U_2 := Matrix([[2y+z^2-1]]);
  combined := MatrixCombine([rc3, rc4], S, [U_1, U_2]);
  V, rc5 := op(combined_1):
  Equations(rc5, S)
```

$$U_1 := \begin{bmatrix} y^2-y \end{bmatrix}$$

$$U_2 := \left[\ 2y + z^2 - 1\ \right]$$

$$combined := \left[\left[\left[\ y^2z^3 + y^2z^2 - 3yz^3 - 3y^2z - 3yz^2 + z^3 + y^2 + 9yz + 2z^2 - y - 3z\ \right], regular\_chain\right]\right]$$

$$[z^4 + z^3 - 3z^2 + z] \qquad \textbf{(4.2.5)}$$

So, the recombination is successful. However, the initial p (with regards to z) of the resulting polynomial is a zero divisor: it vanishes modulo rc_4 and is invertible modulo rc_3. Consequently, the computation of the inverse using the command Inverse splits the combined regular chain rc_5.

```
> p := Initial(V[1,1], R);
  inverse := Inverse(p, rc5, S);
  Equations(inverse[1][1][3], S);
  map(Equations, inverse[2], S)
```

$$p := z^3 + z^2 - 3z + 1$$

$$inverse := [[[1, 1, regular\_chain]], [regular\_chain, regular\_chain]]$$

$$[z]$$

$$[[z - 1], [z^2 + 2z - 1]] \qquad \textbf{(4.2.6)}$$

5. Constructible sets and rational maps

Constructible sets are the geometrical objects naturally attached to triangular decompositions as polynomial ideals are the algebraic concept underlying the computation of Gröbner bases. This relation becomes even more complex and essential in the case of polynomial systems with infinitely many solutions. Basically, constructible sets are what you get when you take algebraic varieties (defined by polynomial ideals) and add the operation of taking complements. More precisely, a constructible set is either a set of points defined by both equations and inequalities, or a finite union of such sets. Note that, in general, the complement of an algebraic variety cannot be described by a polynomial ideal, but it is a constructible set. Constructible sets have the nice property that they are closed under intersections and finite unions, like polynomial ideals, and additionally under complements.

This section presents the ConstructibleSetTools submodule of the RegularChains library. To our knowledge, this is the first general-purpose computer algebra package providing constructible set as a type and exporting a rich collection of operations for manipulating constructible sets. Besides, this module provides routines in support of solving parametric polynomial systems, and several of its commands will be demonstrated in other parts of this document. The examples of the present section illustrate the Theorem of Chevalley which states that the image of a constructible set by a rational map is a constructible set.

5.1 Some high-school examples

Constructible sets appear naturally in many elementary questions from high-school problems. A simple one is the following: for which values of x does $f(x, y) = 0$ have solutions, with f as below?

```
> R := PolynomialRing([y, x]) :
  f := 1 - x·y :
```

The answer is whenever $x \neq 0$ holds. Formally speaking, what we want there is the projection onto the x-axis of the hyperbola $f = 0$. This object is not the zero set of a system of polynomial equations, and therefore it cannot be computed directly via traditional techniques such as a Gröbner basis computation. It requires some finer and more geometrical elimination process. The function Projection implements such a process via triangular decompositions. The output is an object of type *constructible_set*. Its internal representation is given by a list of so called *regular systems*.

```
> cs := Projection({f}, 1, R);
  l := RepresentingRegularSystems(cs, R);
  rs := l[1] :
```

$$cs := constructible\_set$$
$$l := [regular\_system] \qquad \textbf{(5.1.1)}$$

Each regular system is a pair consisting of a regular chain plus one or more inequations:

```
> rc := ConstructibleSetTools[RepresentingChain](rs, R);
  IsEmptyChain(rc, R);
  map(`≠`, RepresentingInequations(rs, R), 0)
```

$$rc := regular\_chain$$
$$true$$
$$[x \neq 0] \qquad \textbf{(5.1.2)}$$

In this example, this regular chain rc is just the empty one and the inequation is just $x \neq 0$. This output may look more complicated than the posed problem itself. So, consider another familiar example but less trivial example: for which values of a, b, c does the equation $a\,x^2 + b\,x + c = 0$ have solutions? This can be seen as another "projection" question.

```
> R := PolynomialRing([x, a, b, c]);
  f := a x^2 + b x + c;
  cs := Projection({f}, 3, R);
  RepresentingRegularSystems(cs, R);
  Display(cs, R);
```

$$R := polynomial\_ring$$
$$f := x^2\,a + x\,b + c$$
$$cs := constructible\_set$$
$$[regular\_system, regular\_system, regular\_system]$$
$$a \neq 0, \left\{ \begin{array}{l} a = 0 \\ b \neq 0 \end{array} \right. , \left\{ \begin{array}{l} a = 0 \\ b = 0 \\ c = 0 \end{array} \right. \qquad \textbf{(5.1.3)}$$

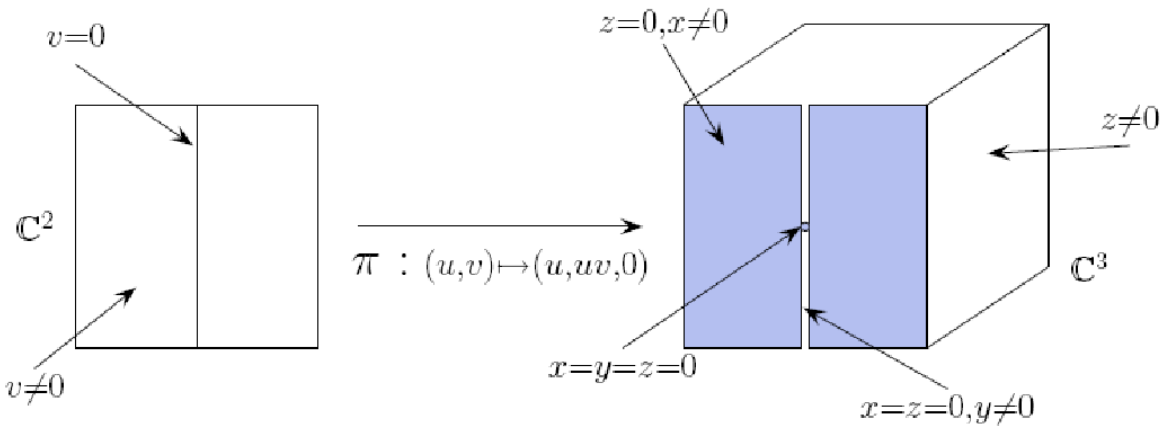
We obtain three regular systems:

(1) $a \neq 0$,

(2) $a = 0,\ b \neq 0,$
(3) $a = b = c = 0.$

5.2 Polynomial map images

The following picture illustrates a polynomial map $\Pi : \mathbb{C}^2 \to \mathbb{C}^3$. The task is to describe the image of this map in $\mathbb{C}^3$. As one can see from the picture, a point with coordinates $(x, y, z) = (0, b, 0)$, where $b \neq 0$, cannot be in the image of Π, whereas any other point with $z = 0$ is. For this reason, the image of Π is a constructible set and not the zero set of a system of polynomial equations. The command PolynomialMapImage below computes the image of Π.




```
> unassign('u','v');
  S := PolynomialRing([u, v]) :
  T := PolynomialRing([x, y, z]) :
```

After specifying the coordinates of the source space S and target space T, the polynomial map is defined and its image is computed.

```
> Π := [u, u v, 0];
  cs := PolynomialMapImage([ ], Π, S, T);
```

$$\Pi := [u,\ u\,v,\ 0]$$

$$cs := constructible\_set \qquad \textbf{(5.2.1)}$$

The constructible set cs is given by a list of two regular systems. The first one is given by $z = 0 \neq x$, and the second one by $x = y = z = 0$.

```
> lrs := RepresentingRegularSystems(cs, T);
  map(Display, lrs, T)
```

$$lrs := [regular\_system,\ regular\_system]$$

$$\left[\begin{Bmatrix} z = 0 \\ x \neq 0 \end{Bmatrix},\ \begin{Bmatrix} x = 0 \\ y = 0 \\ z = 0 \end{Bmatrix}\right] \qquad \textbf{(5.2.2)}$$

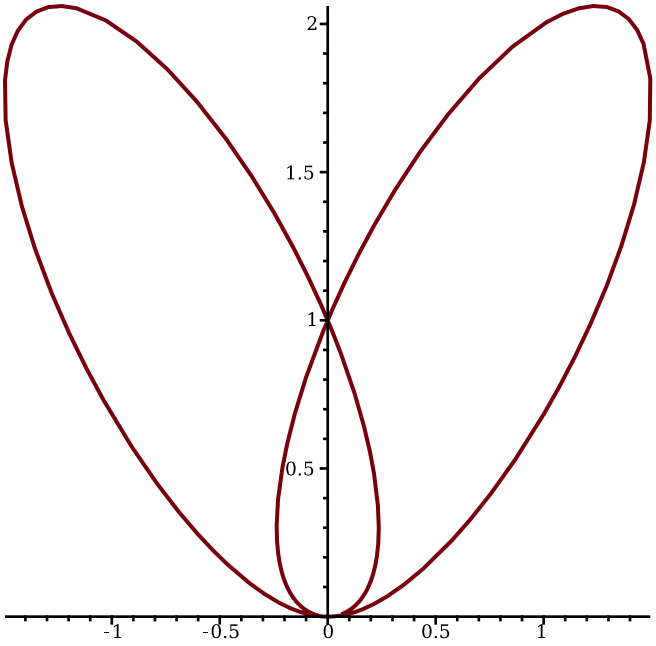
5.3 Rational map images

Images (and pre-images) of constructible sets under rational maps are also supported by the ConstructibleSetTools submodule. To demonstrate this facility, consider the implicit equation of a curve (namely the *tacnode* curve) given by a parametric representation involving rational functions. The parametrization of this curve can be seen as a rational map; its one-dimensional source space S, its two-dimensional target space T, and the image ρ of an arbitrary point are defined below.

```
> S := PolynomialRing([t]):
  T := PolynomialRing([x, y]):
  ρ := [ (t^3 - 6 t^2 + 9 t - 2)/(2 t^4 - 16 t^3 + 40 t^2 - 32 t + 9), (t^2 - 4 t + 4)/(2 t^4 - 16 t^3 + 40 t^2 - 32 t + 9) ]:
```

Here is a parametric plot of the tacnode curve.

```
> plot([op(ρ), t = -10..10]);
```



The parametric explicit representation of a curve is very useful for plotting it. However, in order to answer questions such as "does a given point in target space lie on the curve?", an implicit representation is more useful.
The image of the full one-dimensional space under the rational map ρ is computed below.

```
> F := [ ]:
  cs_1 := RationalMapImage(F, ρ, S, T);
  l := RepresentingRegularSystems(cs_1, T);
  Display(cs_1, T);
```

$$cs_1 := constructible\_set$$

$$l := [regular\_system, regular\_system, regular\_system]$$

$$\left\{\begin{array}{r} 2x^4-3yx^2+y^4-2y^3+y^2=0 \\ 10yx^2+2y^3+2x^2-y^2-y\neq 0 \\ 964x^2y^6-88y^8-480x^2y^5+2104y^7-6858x^2y^4-2316y^6-4328x^2y^3-943y^5-888y^2 \\ y\neq 0 \end{array}\right.$$

$$,\left\{\begin{array}{r} x=0 \\ y-1=0 \end{array}\right. \quad ,\left\{\begin{array}{r} x=0 \\ y=0 \end{array}\right.$$

The result has three components. The second and third ones correspond to the self-intersection points of the tacnode curve, that is, its singularities. The first component l_1 defines all the other points. The role of its inequations is to exclude the two self-intersection points, so that the sets described by the three components are disjoint.

```
> rc := ConstructibleSetTools[RepresentingChain](l1, T):
  eq := Equations(rc, T);
  ineq := ConstructibleSetTools[RepresentingInequations](l1, T)
```

$$eq := [2x^4-3yx^2+y^4-2y^3+y^2]$$

$$ineq := [y,\ 10yx^2+2y^3+2x^2-y^2-y,\ 964x^2y^6-88y^8-480x^2y^5 + 2104y^7-6858x^2y^4-2316y^6-4328x^2y^3-943y^5-888y^2x^2 + 892y^4-72yx^2+318y^3-2x^2+32y^2+y] \qquad \textbf{(5.3.2)}$$

The computations below show that the solution set of the equations of the first component l_1 correspond to the entire tacnode curve. Note the use of the command IsContained which decides whether a constructible set is contained in another.

```
> cs2 := GeneralConstruct(eq, [ ], T);
  IsContained(cs1, cs2, T) and IsContained(cs2, cs1, T)
```

$$cs_2 := constructible\_set$$

$$true \qquad \textbf{(5.3.3)}$$

Therefore, $2x^4-3yx^2+y^2-2y^3+y^4=0$ is the implicit representation of the tacnode curve.

6. Parametric polynomial systems

The ParametricSystemTools module is devoted to solving systems with parameters, including real root classification and complex root classification of such systems, as demonstrated below. The first two examples are dedicated to the complex solutions of parametric polynomial systems. The last, but very detailed, third subsection deals with the real case.

6.1 An example from classical invariant theory

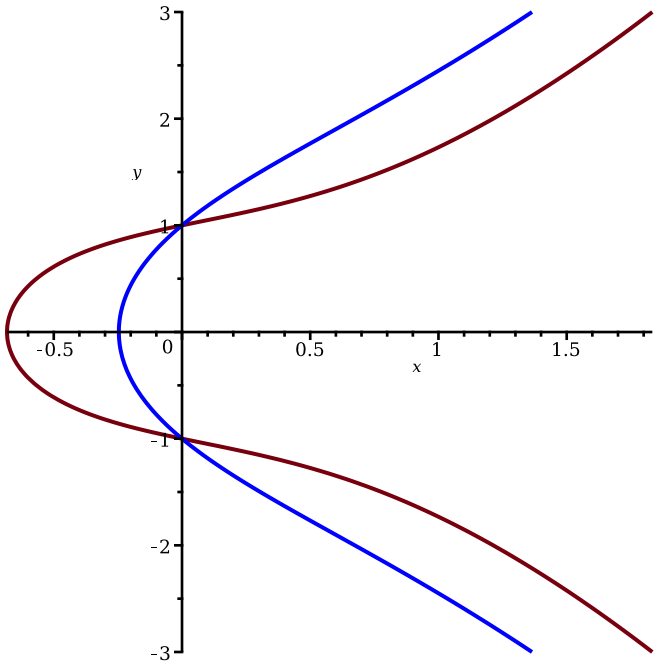
Each of the following polynomials defines an elliptic curve in the complex plane. They depend on parameters a_1 and a_2, respectively. In invariant theory, a classical question is whether there exists a linear fractional map from the first curve to the second one.

```
> f1 := y^2 - x^3 - a1 x - 1;
  f2 := y^2 - x^3 - a2 x - 1
```

$$f_1 := -x^3 - a_1 x + y^2 - 1$$
$$f_2 := -x^3 - a_2 x + y^2 - 1 \qquad \textbf{(6.1.1)}$$

These two curves are plotted below for $a_1 = 1$ and $a_2 = 4$.

```
> with(plots):
  display(implicitplot(f1 | a1 = 1, x = -3..3, y = -3..3), implicitplot(f2
    | a2 = 4, x = -3..3, y = -3..3, color = blue))
```



A generic linear fractional map is applied to the coordinates x and y of the second curve.

```
> unassign('A','B','C','E','F', 'H'):
  f3 := eval(f2, [x = (A x + B y + C)/(G x + H y + K), y = (D x + E y + F)/(G x + H y + K)])
```

$$f_3 := -\frac{(A x + B y + C)^3}{(G x + H y + K)^3} - \frac{(A x + B y + C)\, a_2}{G x + H y + K} + \frac{(D x + E y + F)^2}{(G x + H y + K)^2} - 1 \qquad \textbf{(6.1.2)}$$

Next, one stipulates that the rational function $f_1 - f_3$ must be identically zero. This yields a system of equations *sys*. Without loss of generality, one can assume

that the origin is mapped to the origin, which implies $C = F = 0$.

```
> sys := [coeffs(numer(f_1 - f_3), [x, y]), C, F];
  R := PolynomialRing([B, E, H, A, D, G, K, F, C, a_1, a_2]) :
```

$$sys := \left[-G^3,\ -3G^2H,\ -3G^2K,\ -3GH^2,\ -6GHK,\ -G^3a_1 - 3GK^2,\ -H^3,\ G^3 - 3H^2K,\ -3G^2Ha_1 - 3HK^2,\ AG^2a_2 - 3G^2Ka_1 + A^3 - \mathrm{D}^2G - K^3,\ 3G^2H,\ -3GH^2a_1 + 3G^2K,\ 2AGHa_2 + BG^2a_2 - 6GHKa_1 + 3A^2B - \mathrm{D}^2H - 2\mathrm{D}EG,\ 2AGKa_2 + CG^2a_2 - 3GK^2a_1 + 3A^2C - \mathrm{D}^2K - 2\mathrm{D}FG,\ 3GH^2,\ -H^3a_1 + 6GHK,\ AH^2a_2 + 2BGHa_2 - 3H^2Ka_1 + 3AB^2 - 2\mathrm{D}EH - E^2G + 3GK^2,\ 2AHKa_2 + 2BGKa_2 + 2CGHa_2 - 3HK^2a_1 + 6ABC - 2\mathrm{D}EK - 2\mathrm{D}FH - 2EFG,\ AK^2a_2 + 2CGKa_2 - K^3a_1 + 3AC^2 - 2\mathrm{D}FK - F^2G,\ H^3,\ 3H^2K,\ BH^2a_2 + B^3 - E^2H + 3HK^2,\ 2BHKa_2 + CH^2a_2 + 3B^2C - E^2K - 2EFH + K^3,\ BK^2a_2 + 2CHKa_2 + 3BC^2 - 2EFK - F^2H,\ CK^2a_2 + C^3 - F^2K,\ C,\ F\right] \qquad \textbf{(6.1.3)}$$

To the system of equations *sys*, inequation constraints need to be added. Indeed, the unknowns G, H, and K cannot vanish simultaneously. Hence, the system to be solved consists of the zeros of *sys*, which are not common zeros of these three polynomials. This new system is easily expressed as the set-theoretical difference of two constructible sets.

```
> cs_1 := GeneralConstruct(sys, [ ], R) :
  cs_2 := GeneralConstruct([G, H, K], [ ], R) :
  cs_3 := Difference(cs_1, cs_2, R) :
  Display(cs_3, R)
```

$$\left\{\begin{array}{c} B = 0 \\ E - K = 0 \\ H = 0 \\ a_2A - Ka_1 = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1^2 + a_2a_1 + a_2^2 = 0 \\ K \neq 0 \\ a_2 \neq 0 \end{array}\right., \left\{\begin{array}{c} B = 0 \\ E + K = 0 \\ H = 0 \\ a_2A - Ka_1 = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1^2 + a_2a_1 + a_2^2 = 0 \\ K \neq 0 \\ a_2 \neq 0 \end{array}\right., \left\{\begin{array}{c} B = 0 \\ E - K = 0 \\ H = 0 \\ A - K = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1 - a_2 = 0 \\ K \neq 0 \end{array}\right., \qquad \textbf{(6.1.4)}$$

$$\left\{\begin{array}{c} B=0 \\ E+K=0 \\ H=0 \\ A-K=0 \\ \mathrm{D}=0 \\ G=0 \\ F=0 \\ C=0 \\ a_1-a_2=0 \\ K\neq 0 \end{array}\right\},\ \left\{\begin{array}{c} B=0 \\ E-K=0 \\ H=0 \\ A^2+KA+K^2=0 \\ \mathrm{D}=0 \\ G=0 \\ F=0 \\ C=0 \\ a_1=0 \\ a_2=0 \\ K\neq 0 \end{array}\right\},\ \left\{\begin{array}{c} B=0 \\ E+K=0 \\ H=0 \\ A^2+KA+K^2=0 \\ \mathrm{D}=0 \\ G=0 \\ F=0 \\ C=0 \\ a_1=0 \\ a_2=0 \\ K\neq 0 \end{array}\right\}$$

The question can now be stated in algebraic terms: for which values of the parameters a_1 and a_2 is the constructible set non-empty? Formulas solving for the unknowns are also desired. The command ComprehensiveTriangularize addresses these two requirements. The second argument, 2 in the example below, specifies that the last two indeterminates of the polynomial ring (that is, a_1 and a_2) are the parameters of the system.

```
> ct := ComprehensiveTriangularize(cs[3], 2, R)
```

$$ct := [regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system], [[constructible\_set, [3, 6, 4, 5]], [constructible\_set, [3, 4]], [constructible\_set, [1, 2]]] \quad \textbf{(6.1.5)}$$

The second list returned by this command is a finite partition of the projection of cs onto the parameter space. Each component P of this partition is a constructible set above which the geometrical properties of cs (degree, dimension, etc.) are essentially constant; the corresponding component of cs_3 is given by the regular systems of ct whose indices are associated with P. Below, the three parts of this partition are joined into a single constructible set.

```
> d := map2(op, 1, ct[2]):
  d := Union(d[1], Union(d[2], d[3], R), R):
  Display(d, R);
```

$$\left\{\begin{array}{c} a_1=0 \\ a_2=0 \end{array}\right\},\ \left\{\begin{array}{c} a_1-a_2=0 \\ a_2\neq 0 \end{array}\right\},\ \left\{\begin{array}{c} a_1^2+a_2a_1+a_2^2=0 \\ a_2\neq 0 \end{array}\right\} \quad \textbf{(6.1.6)}$$

Next, check if the answer to the question is the one that is well-known from invariant theory, namely that the two curves can be matched provided that $a_1^3 = a_2^3$.

```
> e := GeneralConstruct([a[1]^3 - a[2]^3], [ ], R);
  `and`(IsContained(d, e, R), IsContained(e, d, R))
```

$$e := constructible\_set$$
$$true \qquad \textbf{(6.1.7)}$$

Finally, the three regular systems (from the first list returned by ComprehensiveTriangularize) are displayed below.

```
> for i to 3 do Display(ct[1][i], R); end do;
```

$$\left\{\begin{array}{c} B = 0 \\ E - K = 0 \\ H = 0 \\ a_2 A - K a_1 = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1^2 + a_2 a_1 + a_2^2 = 0 \\ K \neq 0 \\ a_2 \neq 0 \end{array}\right.$$

$$\left\{\begin{array}{c} B = 0 \\ E + K = 0 \\ H = 0 \\ a_2 A - K a_1 = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1^2 + a_2 a_1 + a_2^2 = 0 \\ K \neq 0 \\ a_2 \neq 0 \end{array}\right.$$

(6.1.8)

$$\begin{cases} B = 0 \\ E - K = 0 \\ H = 0 \\ A - K = 0 \\ \mathrm{D} = 0 \\ G = 0 \\ F = 0 \\ C = 0 \\ a_1 - a_2 = 0 \\ K \neq 0 \end{cases} \quad \textbf{(6.1.8)}$$

6.2 Counting complex solutions

In this short example, it is shown how to determine the number of distinct roots of a generic polynomial equation of degree 4 depending on the parameters a, b, c, d.

```
> unassign('a', 'b', 'c', 'd'):
  R := PolynomialRing([x, a, b, c, d]):
  p := a*x^4+b*x^2+c*x+d;
```

$$p := a\,x^4 + b\,x^2 + c\,x + d \quad \textbf{(6.2.1)}$$

The command ComplexRootClassification is precisely adapted to this purpose. It determines the number of distinct complex solutions of a parametric system (possibly a parametric constructible set) depending on parameters. The second argument, 4 in the example below, indicates that the last four variables of the polynomial ring, a, b, c, d, are the parameters of the equation.

```
> cr := ComplexRootClassification([p], 4, R);
```

$$cr := [[constructible\_set, 1], [constructible\_set, 2], [constructible\_set, 3], [constructible\_set, 4], [constructible\_set, \infty]] \quad \textbf{(6.2.2)}$$

The result is a partition of the parameter space into five sets: those parameter values for which p has exactly 1, 2, 3, or 4 complex roots, respectively, and those parameter values for which p has infinitely many roots (which happens only if $a = b = c = d = 0$).

Suppose you are interested in the case of three distinct roots. Display the equations and inequations of the regular systems defining the constructible set, giving the necessary and sufficient conditions for the polynomial p to have three distinct complex roots.

```
> Info(cr[3][1], R);
```

$$[[256\,d^3\,a^2 + (-128\,d^2\,b^2 + 144\,c^2\,d\,b - 27\,c^4)\,a + 16\,b^4\,d - 4\,b^3\,c^2], [d, a, (8\,d\,b - 9\,c^2)\,a - 2\,b^3, c, 32\,d\,b - 9\,c^2, (1073741824\,b^{10}\,d^{10} - 66437775360\,b^9\,c^2\,d^9 + 672682475520\,b^8\,c^4\,d^8 - 2623461654528\,c^6\,d^7\,b^7 + 5164940132352\,c^8\,d^6\,b^6 \quad \textbf{(6.2.3)}$$

$$-5810557648896\,c^{10}d^5b^5+3961743851520\,c^{12}d^4b^4$$
$$-1665238487040\,c^{14}d^3b^3+421513492032\,c^{16}d^2b^2$$
$$-58887914328\,c^{18}d\,b+3486784401\,c^{20})\,a-268435456\,b^{12}d^9$$
$$+12683575296\,b^{11}c^2d^8-98524200960\,b^{10}c^4d^7$$
$$+294298583040\,b^9c^6d^6-439871275008\,b^8c^8d^5$$
$$+368924295168\,b^7c^{10}d^4-181579926528\,b^6c^{12}d^3$$
$$+52038702720\,b^5c^{14}d^2-8035387920\,b^4c^{16}d+516560652\,b^3c^{18}]],$$
$$[[27\,c^2a+4\,b^3,d],[c,b]],[[c,d],[b,a]]$$

This output is to be interpreted as follows. There are three cases:

- $c=d=0$ and $a\neq 0\neq b$. In this case, the polynomial degenerates to $p=a\,x^4+bx^2$, and the three roots are a double root at $x=0$ and two distinct simple roots at $\pm\sqrt{-\frac{b}{a}}$.
- $4\,b^3+27\,a\,c^2=0=d$ and $b\neq 0\neq c$. In this case, the polynomial degenerates to $p=a\,x^4+b\,x^2+c\,x$, which has $x=0$ as a simple root. The expression $4\,b^3+27\,a\,c^2$ is the discriminant of the cofactor $\frac{p}{x}=a\,x^3+b\,x+c$, and the condition that it must be zero implies that this polynomial has at least a double root. Finally, the condition $b\neq 0$ ensures that the latter polynomial does not have a triple root, that is, it has one double and one single root.
- $256\,d^3a^2+(-128\,d^2b^2+144\,d\,c^2b-27\,c^4)\,a+16\,d\,b^4-4\,c^2b^3=0$ and none of the polynomials in the last list are zero, including $a\neq 0, c\neq 0, d\neq 0$, $32\,b\,d-9\,c^2\neq 0$, and $2\,b^3-(8\,b\,d-9\,c^2)a\neq 0$. The first polynomial is a factor of the discriminant of p:

```
> discrim(p, x);
```

$$a\,(256\,d^3a^2-128\,a\,b^2d^2+144\,a\,b\,c^2d-27\,a\,c^4+16\,b^4d-4\,b^3c^2) \qquad \textbf{(6.2.4)}$$

The condition that it must be zero implies that p has at least a double root, and the remaining inequations ensure that all the other roots are simple.

6.3 Real root classification and border polynomial

Real root classification

A *semi-algebraic system* (SAS) is a polynomial system containing polynomial equations $p=0$, inequations $p\neq 0$, and inequalities $p>0$ or $p\geq 0$. If a semi-algebraic system contains parameters, it is called a parametric SAS. For a parametric SAS and a prescribed non-negative integer n, the *real root classification problem* is to compute conditions on the parameters such that the system has exactly n distinct real solutions. In the RegularChains package, a semi-algebraic system is given by four lists F, N, P, H of polynomials, representing the equations, non-negativity conditions (weak inequalities), positivity conditions (strict inequalities), and equations, respectively. The RealRootClassification command takes these four lists as the first four arguments. As in the examples above, the fifth argument, d, is an integer specifying that the last d indeterminates of the underlying polynomial ring are

the parameters of the system. The sixth argument, n, specifies the desired number of distinct real solutions, and the last argument is the polynomial ring R. For example,

```
> R := PolynomialRing([x, a, b, c]):
  F := [x^2*a+x*b+c]:
  N := []:P := [x]:H := [a]:
  rr := RealRootClassification(F, N, P, H, 3, 2, R);
```

$$rr := [[regular\_semi\_algebraic\_set], border\_polynomial] \qquad \textbf{(6.3.1.1)}$$

Here, a, b, c are viewed as parameters and the task is to find conditions for the general polynomial of degree 2 to have exactly 2 distinct positive real solutions.

The output is a list, rr_1 of *regular semi-algebraic sets* and a *border polynomial object* (BP), rr_2; together these describe a first-order logic formula. More precisely, the list rr_1 gives necessary and sufficient conditions for the input system to have the prescribed number of real solutions, provided that the condition encoded by the BP holds. The role of the border polynomial is to exclude degenerate cases, which can be handled later with further computations (more on this later).

The command below shows the contents of the BP, which is actually a list of polynomials. The logical condition encoded by the BP is that none of those polynomials should be zero.

```
> Info(rr[2], R);
```

$$[a, b, c, -4\,a\,c + b^2] \qquad \textbf{(6.3.1.2)}$$

The following commands extract information about the list of regular SAS and display the logical condition that these encode, using the command Display.

```
> ss := rr[1][1];
  Display(ss, R);
```

$$ss := regular\_semi\_algebraic\_set$$

$$\begin{cases} c < 0 \textbf{ and } a < 0 \textbf{ and } b > 0 \textbf{ and } 4\,a\,c - b^2 < 0 \\ \textbf{or } c > 0 \textbf{ and } a > 0 \textbf{ and } b < 0 \textbf{ and } 4\,a\,c - b^2 < 0 \end{cases} \qquad \textbf{(6.3.1.3)}$$

The output of the last command is to be interpreted as the logical disjunction ("or") of the conditions in the last two rows; each row represents the logical conjunction ("and") of the individual inequalities.

As a first approximation, a regular SAS is a subset of the real solutions of a regular chain. More precisely, it is encoded by three pieces of data:

- a *quantifier free formula* qf,
- a regular chain rc, and
- a list of lists of indices l.

This encoding is called a *parametric box*. The formula qf specifies the conditions on the variables which are free in rc (non-algebraic), whereas rc and l specify the conditions on the variables which are algebraic in rc.

In the example above, rc and l are empty (which is often the case); therefore, the conditions encoded by the parametric box are just the ones of qf.

```
> rc := RepresentingChain(ss, R);
```

```
Equations(rc, R);
box := RepresentingBox(ss);
l := RepresentingRootIndex(box);
```

$$rc := regular\_chain$$

$$[\,]$$

$$box := parametric\_box$$

$$l := [\,] \qquad \textbf{(6.3.1.4)}$$

Summarizing, the answer to the particular question is: provided that the BP holds, the input SAS has exactly 2 distinct positive real solutions if and only if qf holds. Note that, in the example, the conditions of qf imply those of the BP; thus the answer can be simplified further as follows:
the input SAS has exactly 2 distinct positive real solutions if and only if qf holds.

Setting infolevel to a nonzero value also provides information on how to interpret the above result.

```
> infolevel[RegularChains] := 1:
  RealRootClassification(F, N, P, H, 3, 2, R);
  infolevel[RegularChains] := 0:
TRDtofind1: FINAL RESULT:
TRDtofind1: The system has given number of real solution(s) IF AND ONLY
IF
TRDtofind1: [R[1] < 0 R[2] < 0 0 <= R[3] R[4] < 0]
TRDtofind1: OR
TRDtofind1: [0 < R[1] 0 < R[2] R[3] <= 0 R[4] < 0]
TRDtofind1: where
TRDtofind1: R[1] = c
TRDtofind1: R[2] = a
TRDtofind1: R[3] = b
TRDtofind1: R[4] = 4*a*c-b^2
TRDtofind1: PROVIDED THAT

TRDtofind1: c <> 0
TRDtofind1: a <> 0
TRDtofind1: 4*a*c-b^2 <> 0
TRDtofind1: 0.4e-2*seconds
```

$$[[regular\_semi\_algebraic\_set], border\_polynomial] \qquad \textbf{(6.3.1.5)}$$

There are two alternative ways to obtain the answer in a simpler (with less commands) but more coarse manner. The first one, shown above, is to set the infolevel to 1. The second one, shown below, is to use the Info command on the regular SAS. With this second approach, one can directly see the encoding of the regular SAS, that is, the parametric box; however, the conditions defining the regular SAS are not shown explicitly.

```
> Info(ss, R);
```

$$\left[\left[\left[c, a, b, 4\,a\,c - b^2\right]\right], [[-1, -1, 1, -1], [1, 1, -1, -1]]\right], [\,], [\,]\right] \qquad \textbf{(6.3.1.6)}$$

More about border polynomials

The border polynomial is actually a list of polynomials such that the union of the hypersurfaces given by these polynomials contains all the "abnormal" (non-generic) parameter values of the parametric input SAS. For the previous example, the border polynomial is $[c, a, 4\,a\,c - b^2]$. It is intuitive that the zeroes of these polynomials represent degenerate cases: the first two indicate vanishing of the trailing and leading coefficients, respectively, and the last polynomial is

the discriminant of the input polynomial. You can use the Info command to view the contents of the BP:

```
> Info(rr[2], R);
```

$$[a, b, c, -4\,a\,c + b^2] \qquad \textbf{(6.3.2.1)}$$

You might want to go further in analyzing the number of real solutions of the input SAS; that is, you might want to figure out what the answer is in these degenerate cases. This is very simple: it suffices to add the product of the polynomials in the BP to the list of equations when calling RealRootClassification (or to add the polynomials in the BP one by one to the equations of the system and call RealRootClassification accordingly).

In the example above, the question was for which parameter values does the input system has 2 distinct positive real solutions. When adding the polynomial $4\,a\,c - b^2$ to the input system, below, you see that the list of regular SAS in the output from RealRootClassification is empty. This just means that it is impossible that the equation $a\,x^2 + b\,x + c = 0$ has 2 distinct positive real solutions when its discriminant is zero.

```
> rr := RealRootClassification([4*a*c-b^2, op(F)], N, P, H, 3, 2,
  R);
  bp := rr[2];Info(bp, R);
```

$$rr := [[\,], border\_polynomial]$$

$$bp := border\_polynomial$$

$$[b, a] \qquad \textbf{(6.3.2.2)}$$

The above output means: provided that $b \neq 0 \neq c$ holds, the new system never admits two distinct positive real solutions.

More about regular semi-algebraic sets

A regular semi-algebraic set is essentially a subset of the real solutions of a regular chain. When this subset is finite, it can be encoded as a list of points with real coordinates. Each of these real coordinates must be an algebraic number defined by a regular chain and an isolating interval. Such an encoding is called a *numerical box*.

When this subset is not finite, the idea is to use a regular chain rc that is not zero-dimensional. Such a regular chain will have variables which are free in rc, that is, variables that do not belong to the main variables of the polynomials defining rc. Those free variables can be regarded as parameters. For rc to have real solutions, those free variables may need to satisfy some inequalities; this is where the quantifier-free formula qf comes in. More technically, qf defines an open semi-algebraic set (SAS) at every point of which rc specializes well and separates well (again, viewing the free variables of rc as parameters). This technical condition has the following important consequence: the number of real roots of rc is constant on the open SAS; moreover, each root of rc (complex or real) is a simple root on the open SAS. This in turn has the following consequence: the real solutions of rc can be indexed uniformly on the open SAS. Now this is where the list of lists of indices l comes in; it selects some of the real roots of rc.

To illustrate the concept of a regular semi-algebraic set in a sufficiently general situation, the original question will be modified as follows: Determine under which conditions on a, b, c, d the equation $x^2 + d = 0$ has 2 distinct real solutions subject to d satisfying $a\,d^2 + b\,d + c = 0$. Note that d plays a special role among the parameters, so it is ordered before a, b, c.

```
> F := [x^2 + d, a*d^2 + b*d + c]:
  N := [ ]:
  P := [ ]:
  H := [ ]:
  R := PolynomialRing([x, d, a, b, c]):
  rr := RealRootClassification(F, N, P, H, 4, 2, R)
```

$$rr := [[regular\_semi\_algebraic\_set], border\_polynomial] \qquad \textbf{(6.3.3.1)}$$

Recall that the output consists of a list of regular SAS and a BP object. The command RepresentingBox applied to a regular SAS returns its encoding as either a numerical box or a parametric box; the command IsParametricBox will determine which of these two cases holds.

```
> {seq(IsParametricBox(RepresentingBox(ss, R)), ss = rr[1])}
```

$$\{true\} \qquad \textbf{(6.3.3.2)}$$

Display the BP using the Info command.

```
> Info(rr[2], R)
```

$$[d] \qquad \textbf{(6.3.3.3)}$$

Select the first parametric box for further inspection. The command Info shows the raw data defining the parametric box, whereas the command Display pretty prints the conditions encoded by this data.

```
> ss := rr[1][1]:
  box := RepresentingBox(ss, R);
  Info(box, R);
  Display(box, R);
```

$$box := parametric\_box$$

$$[[[d], [[-1]]], [a\,d^2 + b\,d + c], [[1]]]$$

$$\begin{cases} c = -a\,d^2 - b\,d \\ d < 0 \end{cases} \qquad \textbf{(6.3.3.4)}$$

The three pieces of data defining a parametric box can also be accessed independently as shown below. Note that RepresentingChain applies directly to a regular SAS since both numerical boxes and parametric boxes have a representing regular chain. The components qf and l, however, are specific to parametric boxes.

(Note that objects of types regular_chain, regular_system, constructible_set, quantifier_free_formula, parametric_box, regular_semi_algebraic_set, and border_polynomial can all be printed by the Info and Display command.)

```
> box := RepresentingBox(ss, R);
  rc := RepresentingChain(ss, R);
```

```
qf := RepresentingQuantifierFreeFormula(box);
l := RepresentingRootIndex(box);
Display(rc, R);
Display(qf, R);
Display(box, R)
```

$$box := parametric\_box$$
$$rc := regular\_chain$$
$$qf := quantifier\_free\_formula$$
$$l := [[1]]$$
$$a\,d^2 + b\,d + c = 0$$
$$d < 0$$
$$\begin{cases} c = -a\,d^2 - b\,d \\ d < 0 \end{cases} \qquad \textbf{(6.3.3.5)}$$

An advanced example of real root classification

Define a polynomial ring.

```
> unassign('p','q');
  R := PolynomialRing([x, y, z, q, p]) :
```

Consider the following equations...

```
> g1 := y^2 + z^2 - 2 y z p - 1 :
  g2 := z^2 + x^2 - 2 z x q - 1 :
  g3 := x^2 + y^2 - 2 x y q - 1 :
  g4 := 5 q^2 - 1 :
  F := [g1, g2, g3, g4] :
```

...and constraints.

```
> N := [ ] :
  P := [x, y, z, 1 - p^2, 1 - q^2, p + 1 - 2 q^2] :
  H := [ ] :
```

The last 2 unknowns, p and q, are viewed as parameters. Suppose you want to know the conditions for the system to have no real solutions.

```
> rr := RealRootClassification(F, N, P, H, 2, 0, R)
```

$$rr := [[regular\_semi\_algebraic\_set, regular\_semi\_algebraic\_set], border\_polynomial] \qquad \textbf{(6.3.4.1)}$$

```
> Display(rr1, R);
  Display(rr2, R)
```

$$\left[\begin{cases} q = RootOf\left(5\_Z^2 - 1, index = real_1\right) \\ p > 0 \textbf{ and } 2p - 1 > 0 \textbf{ and } 5p - 3 \neq 0 \textbf{ and } 5p + 3 > 0 \textbf{ and } 5p^2 - 1 > 0 \end{cases}, \begin{cases} q = RootOf\left(5\_Z^2 - 1, index = real_2\right) \\ p > 0 \textbf{ and } 2p - 1 > 0 \textbf{ and } 5p - 3 > 0 \textbf{ and } 5p + 3 > 0 \textbf{ and } 5p^2 - 1 > 0 \end{cases}\right.$$

$$[p, 2p-1, 5p-3, 5p+3, 2p+1, 5p^2-1] \qquad \textbf{(6.3.4.2)}$$

The output should be read as follows: provided that not one of the polynomials in the BP (displayed in the last row) vanishes, the input system has no real solutions if and only if one of the following two conditions holds:

(1) $p > 0 \wedge 5p+3 > 0 \wedge 2p-1 > 0 \wedge 5p^2-1 > 0 \wedge q$ is the first (from left to right) root of $5q^2-1$;

(2) $p > 0 \wedge 5p-3 > 0 \wedge 5p+3 > 0 \wedge 2p-1 > 0 \wedge 5p^2-1 > 0 \wedge q$ is the second root of $5q^2-1$.

Why do border polynomials form a type?

Consider again the parametric system in one variable and three parameters, associated with a generic univariate polynomial equation of degree 2.

```
> R := PolynomialRing([x, a, b, c]):
  F := [a x^2 + b x + c]:
  N := []:
  P := []:
  H := []:
```

The border polynomial is intrinsically associated with this system, independent of the prescribed number of real solutions of interest. The BP can be computed directly by the BorderPolynomial command.

```
> bp := BorderPolynomial(F, N, P, H, 3, R):
  Info(bp, R)
```

$$[a, -4ac+b^2] \qquad \textbf{(6.3.5.1)}$$

The reason why border polynomials form a type in the RegularChains package (and cannot just be viewed as lists of polynomials) is that under special circumstances, the border polynomial of a parametric semi-algebraic system may degenerate.

The first such case is when the parameters do not appear in the system of polynomials; then the border polynomial is [1], as in the next example.

```
> bp := BorderPolynomial([x^2 - 1], [], [], [], 3, R):
  eval(bp);
  Info(bp, R)
```

table([*PolynomialList* = [1], *type* = *border_polynomial*, *PolynomialRing* = *polynomial_ring*])

$$[1] \qquad \textbf{(6.3.5.2)}$$

Another special circumstance is that of overdetermined or inconsistent systems; then the border polynomial is denoted as [ ], as in the example below.

```
> F := [a x^2 + b x + c, a, b]:
  bp := BorderPolynomial(F, [], [], [], 1, R):
  eval(bp);
  Info(bp, R)
```

table([*PolynomialList* = [ ], *type* = *border_polynomial*, *PolynomialRing* = *polynomial_ring*])

$$[\,] \qquad \textbf{(6.3.5.3)}$$

Finally, the last special case is when the input system has "generically" infinitely many complex solutions;
then the border polynomial is denoted as $[0]$, as in the example below (this is because only two of the variables, b and c, are considered as parameters).

```
> F := [a x^2 + b x + c]:
  bp := BorderPolynomial(F, [ ], [ ], [ ], 2, R) :
  eval(bp);
  Info(bp, R)
```

table([*PolynomialList* = [0], *type* = *border_polynomial*, *PolynomialRing* = *polynomial_ring*])

$$[0] \qquad \textbf{(6.3.5.4)}$$

6.4 Solving parametric semi-algebraic systems with application to the study stability of biological systems

The biological system is described by the following system of differential equations.
Its right hand side encodes the equilibria:

```
> ode := {diff(x(t),t) = -x(t)+s/(1+y(t)^2), diff(y(t),t)=-y(t)+s/
  (1+x(t)^2)};
  F := [-x+s/(1+y^2), -y+s/(1+x^2)];
```

$$ode := \left\{ \frac{\mathrm{d}}{\mathrm{d}t}\, x(t) = -x(t) + \frac{s}{1+y(t)^2},\ \frac{\mathrm{d}}{\mathrm{d}t}\, y(t) = -y(t) + \frac{s}{1+x(t)^2} \right\}$$

$$F := \left[-x + \frac{s}{y^2+1},\ -y + \frac{s}{x^2+1} \right] \qquad \textbf{(6.4.1)}$$

The following two Hurwitz determinants determine the stability of hyperbolic equilibria:

```
> D1 := -(diff(F[1],x)+diff(F[2],y)); #D1 is 2
  D2 := diff(F[1],x)*diff(F[2],y)-diff(F[1],y)*diff(F[2],x);
```

$$D1 := 2$$

$$D2 := 1 - \frac{4 s^2 y x}{\left(y^2+1\right)^2 \left(x^2+1\right)^2} \qquad \textbf{(6.4.2)}$$

The semi-algebraic system below encodes the asymtotically stable hyperbolic equilibria:

```
> P := [numer(normal(F[1]))=0, numer(normal(F[2]))=0, x>0, y>0,
  s>0, numer(D2)>0];
```

$$P := [-x y^2 + s - x = 0, -x^2 y + s - y = 0, 0 < x, 0 < y, 0 < s, 0 < x^4 y^4 + 2 x^4 y^2 + 2 x^2 y^4 - 4 s^2 y x + x^4 + 4 x^2 y^2 + y^4 + 2 x^2 + 2 y^2 + 1] \quad \textbf{(6.4.3)}$$

Compute a real comprehensive triangular decomposition of **P** w.r.t. the parameter ***s***:

```
> R := PolynomialRing([y, x, s]):
  ctd := RealComprehensiveTriangularize(P, 1, R);
```

$$ctd := [[[1, squarefree\_semi\_algebraic\_system], [2, squarefree\_semi\_algebraic\_system]], [[semi\_algebraic\_set, [\,]], [semi\_algebraic\_set, [1]], [semi\_algebraic\_set, [2]]]] \quad \textbf{(6.4.4)}$$

Derive the values of ***s*** s.t. **P** has **2** positive real solutions, that is the biological system is bistable:

```
> ctd2 := RealComprehensiveTriangularize(ctd, R, 2);
  Display(ctd2[2][1][1],R);
```

$$ctd2 := [[[1, squarefree\_semi\_algebraic\_system]], [[semi\_algebraic\_set, [1]]]]$$

$$[2 < s] \quad \textbf{(6.4.5)}$$

The two asymptotically stable equilibria are represented by a *squarefree_semi_algebraic_system*.

```
> ss := ctd2[1][1][2];
  Display(ss, R);
```

$$ss := squarefree\_semi\_algebraic\_system$$

$$\begin{cases} x y - 1 = 0 \\ x^2 - s x + 1 = 0 \\ y > 0 \\ x > 0 \\ s^7 x - s^6 - 6 x s^5 + 5 s^4 + 8 x s^3 - 4 s^2 > 0 \end{cases} \quad \textbf{(6.4.6)}$$

7. FFT-based polynomial arithmetic

The module FastArithmeticTools supports the implementation of modular methods for computing with polynomials, algebraic extensions, and thus regular chains. This support consists of fundamental operations such as resultant,

polynomial gcds, normal forms, etc.

The commands of this module work in prime characteristic and rely on asymptotically fast algorithms. Most of the underlying polynomial arithmetic is performed by C code and relies on (multi-dimensional) Fast Fourier Transform (FFT) and straight line programs (SLPs). This C code is highly optimized and implements the Truncated Fourier Transform (TFT) and an improved version of Montgomery's trick.

- The commands IteratedResultantDim0 and IteratedResultantDim1 compute the iterated resultant of a polynomial with regards to a regular chain of dimension 0 and 1, respectively.
- The commands NormalFormDim0 and ReduceCoefficientsDim0 compute the normal form of a polynomial with regards to a zero-dimensional regular chain.
- The commands NormalizePolynomialDim0 and NormalizeRegularChainDim0 normalize a polynomial (with regards to a zero-dimensional regular chain) and a regular chain (with regards to itself).
- The command RegularizeDim0 tests whether a polynomial is invertible modulo a zero-dimensional regular chain.
- The commands RegularGcdBySpecializationCube, ResultantBySpecializationCube, and SubresultantChainSpecializationCube compute resultants and polynomial gcds modulo a regular chain using fast evaluation and interpolation.
- The commands RandomRegularChainDim0 and RandomRegularChainDim1 compute random regular chains of given degrees.
- Finally, the command BivariateModularTriangularize solves bivariate polynomial systems.

Most of the commands of FastArithmeticTools implements core operations on regular chains such as regularity test and polynomial gcd modulo a regular chain. However, these commands have several constraints. On top of the characteristic constraint (detailed below), the current regular chain must have dimension zero or one. There is only one exception: the command RegularGcdBySpecializationCube requires no assumption about the dimension. Note also that some commands do not take any regular chains as input (for instance, SubresultantChainSpecializationCube and ResultantBySpecializationCube).

Since FastArithmeticTools relies heavily on direct FFT and Montgomery's trick, the characteristic of the polynomial ring must be a prime number p satisfying the following properties. First, it should not be greater than 962592769. Secondly, the number $p-1$ should be divisible by a sufficiently large power of 2. The power 2^{20} is often sufficient. If this power of 2 is not large enough, then an appropriate error message is returned. Using try-catch statements is highly recommended when programming with the commands of this submodule.

7.1 The impact of fast arithmetic

The purpose of the session below is to demonstrate how resultants can be

computed with the module FastArithmeticTools.

```
> restart;
  with(RegularChains);
  with(FastArithmeticTools);
```

[*AlgebraicGeometryTools, ChainTools, ConstructibleSetTools, Display, DisplayPolynomialRing, Equations, ExtendedRegularGcd, FastArithmeticTools, Inequations, Info, Initial, Intersect, Inverse, IsRegular, LazyRealTriangularize, MainDegree, MainVariable, MatrixCombine, MatrixTools, NormalForm, ParametricSystemTools, PolynomialRing, Rank, RealTriangularize, RegularGcd, RegularizeInitial, SamplePoints, SemiAlgebraicSetTools, Separant, SparsePseudoRemainder, SuggestVariableOrder, TRDFM_elim_eqsfirst, TRDconvex_union, Tail, Triangularize*]

[*BivariateModularTriangularize, IteratedResultantDim0, IteratedResultantDim1, NormalFormDim0, NormalizePolynomialDim0, NormalizeRegularChainDim0, RandomRegularChainDim0, RandomRegularChainDim1, ReduceCoefficientsDim0, RegularGcdBySpecializationCube, RegularizeDim0, ResultantBySpecializationCube, SubresultantChainSpecializationCube*] **(7.1.1)**

Set the polynomial ring.

```
> p := 962592769;
  vars := [a, b];
  R := PolynomialRing(vars, p);
```

$$p := 962592769$$
$$vars := [a, b]$$
$$R := polynomial\_ring \quad \textbf{(7.1.2)}$$

Define the input polynomials.

```
> degbound := 40;
  f[1] := randpoly(vars, dense, degree = degbound) mod p:
  f[2] := randpoly(vars, dense, degree = degbound) mod p:
```

$$degbound := 40 \quad \textbf{(7.1.3)}$$

Evaluate/interpolate by subproduct tree techniques.

```
> t[1] := time( ):
  SCube := SubresultantChainSpecializationCube(f[1], f[2], a, R, 0);
  r[1] := ResultantBySpecializationCube(f[1], f[2], a, SCube, R):
  t[1] := time( ) - t[1]
```

$$SCube := subresultant\_chain\_specialization\_cube$$
$$t_1 := 0.476 \quad \textbf{(7.1.4)}$$

Evaluate/interpolate by multi-dimensional TFT.

```
> t2 := time( ) :
  SCube := SubresultantChainSpecializationCube(f1, f2, a, R, 1); r2 :=
     ResultantBySpecializationCube(f1, f2, a, SCube, R) :
  t2 := time( ) − t2
```

$$SCube := subresultant\_chain\_specialization\_cube$$
$$t_2 := 0.218 \qquad (7.1.5)$$

Evaluate/interpolate without fast arithmetic.

```
> t3 := time( ) :
  r3 := Resultant(f1, f2, a) mod p :
  t3 := time( ) − t3
```

$$t_3 := 10.519 \qquad (7.1.6)$$

Compare the results.

```
> evalb(`and`(r1 = r2, r2 = r3));
  degree(r1);
   t1, t2, t3;
```

$$true$$
$$1600$$
$$0.476, 0.218, 10.519 \qquad (7.1.7)$$

7.2 The impact of modular methods together with fast arithmetic

The session below shows how polynomial gcds modulo regular chains can be computed with the module FastArithmeticTools. This applies also to polynomial gcds over towers of field extensions.

```
> restart:
  with(RegularChains):
  with(FastArithmeticTools):
  with(ChainTools):
```

Set the polynomial ring.

```
> p := 962592769:
  vars := [a, b, c]:
  R := PolynomialRing(vars, p):
```

Define the polynomials.

```
> degbound := 7;
  f1 := randpoly(vars, dense, degree = degbound) mod p :
  f2 := randpoly(vars, dense, degree = degbound) mod p :
```

$$degbound := 7 \qquad (7.2.1)$$

Evaluate/interpolate by multi-dimensional TFT.

```
> t2 := time( ) :
  SCube := SubresultantChainSpecializationCube(f[1], f[2], a, R, 1) :
  r2 := ResultantBySpecializationCube(f[1], f[2], a, SCube, R) :
  rc := Chain([r2], Empty(R), R) :
  g2 := RegularGcdBySpecializationCube(f[1], f[2], rc, SCube, R) :
  t2 := time( ) − t2
```

$$t_2 := 0.387 \qquad (7.2.2)$$

Compute without fast arithmetic and without modular methods (evaluation / interpolation). Since the command RegularGcd tries to use modular methods whenever possible, change the characteristic of the ring to a small prime, so as to enforce the use of non-modular and non-FFT-based algorithms.

```
> t3 := time( ) :
  S := PolynomialRing(vars, 257) :
   r3 := Resultant(f1, f2, a) mod p :
  rc := Chain([r3], Empty(S), S) :
  g3 := RegularGcd(f1, f2, a, rc, S) :
  t := time( ) − t3
```

$$t := 10.937 \qquad (7.2.3)$$

7.3 Accelerating the core operations of the RegularChains library

Testing whether a polynomial is invertible modulo (the saturated ideal of) a zero-dimensional regular chain is a fundamental operation. Its implementation RegularizeDim0 in the FastArithmeticTools takes advantage of FFT-based polynomial arithmetic and improves on the command Regularize. Note that this latter command implements a more general algorithm (with no assumptions on characteristic or dimension).

```
> restart:
  with(RegularChains):
  with(FastArithmeticTools):
  with(ChainTools):
```

Set the polynomial ring.

```
> p := 962592769:
  vars := [x1, x2, x3, x4]:
  R := PolynomialRing(vars, p):
```

Define a random (dense) regular chain and a polynomial.

```
> N := nops(vars):
  d := 5;
  degrees := [3, 4, 5, 8];
  rc := RandomRegularChainDim0(vars, degrees, p):
  p := `mod`(`mod`(randpoly(vars, dense, degree = d)+rand(), p), p)
  :
```

$$d := 5$$
$$degrees := [3, 4, 5, 8] \quad \textbf{(7.3.1)}$$

Compute with the modular code.

```
> t1 := time( ) :
  RegularizeDim0(p, rc, R) :
  t1 := time( ) − t1
```

$$t_1 := 0.167 \quad \textbf{(7.3.2)}$$

Compute with the generic algorithm (without modular methods and without asymptotically fast arithmetic). Here again, change the characteristic to prevent use of the fast code.

```
> t2 := time( ) :
  Regularize(p, rc, PolynomialRing(vars, 17));
  t2 := time( ) − t2
```

$$[[rc], [\,]]$$
$$t_2 := 0.422 \quad \textbf{(7.3.3)}$$

7.4 Solving large bivariate systems

In this example, solve a dense bivariate and square system which has 2500 solutions.

```
> restart:
  with(RegularChains):
  with(FastArithmeticTools):
  with(ChainTools):
```

Set the polynomial ring.

```
> p := 469762049:
  vars := [x, y]:
  R := PolynomialRing(vars, p):
```

Define the polynomials and inspect their number of terms.

```
> degbound := 50 :
   f1 := randpoly(vars, dense, degree = degbound) mod p :
  f2 := randpoly(vars, dense, degree = degbound) mod p :
  nops(f1); nops(f2)
                                  1314
                                  1321                          (7.4.1)
```

Solve the system given by $f_1 = f_2 = 0$ using fast arithmetic.

```
> t := time( ) :
  l := BivariateModularTriangularize([f1, f2], R) :
  t := time( ) − t
                              t := 0.691                        (7.4.2)
```

Check the number of solutions and its number of terms.

```
> map(NumberOfSolutions, l, R);
  map(nops, Equations(l[1], R));
                                 [2500]
                               [2404, 2501]                     (7.4.3)
>
```

8. Cylindrical Algebraic Decomposition and Quantifier Elimination

The RegularChains library provides a set of commands for computing cylindrical algebraic decomposition (CAD, see command CylindricalAlgebraicDecompose) and doing quantifier elimination (QE, see command QuantifierElimination). The underlying algorithm first computes a cylindrical decomposition of the complex space (CCD, see command CylindricalDecompose), which is futher refined into a cylindrical algebraic decomposition of the real space. Such an algorithm is different from the traditional projection-lifting algorithm introduced by George Collins and further refined by others.

8.1 Cylindrical decomposition of the complex space

A CCD is a partition of the complex space into disjoint cells such that they are cylindrically arranged,
meaning the projection of any two cells onto any lower dimensional space are either idential or disjoint,
and each cell is the zero set of a regular system.

```
> R := PolynomialRing([y, x]);
  F := [x^2+y^2-1];
```

$$R := polynomial\_ring$$
$$F := [x^2 + y^2 - 1] \qquad \textbf{(8.1.1)}$$

```
> ccd := CylindricalDecompose(F, R);
  Display(ccd, R);
```

$$ccd := c\_c\_d$$

$$\left[\left\{\begin{array}{l} y = 0 \\ x - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x - 1 = 0 \\ y \neq 0 \end{array}\right., \left\{\begin{array}{l} y = 0 \\ x + 1 = 0 \end{array}\right., \left\{\begin{array}{l} x + 1 = 0 \\ y \neq 0 \end{array}\right., \left\{\begin{array}{l} y^2 + x^2 - 1 = 0 \\ x^2 - 1 \neq 0 \end{array}\right., \left\{\begin{array}{l} y^2 + x^2 - 1 \neq 0 \\ x^2 - 1 \neq 0 \end{array}\right.\right] \qquad \textbf{(8.1.2)}$$

```
> ccd := CylindricalDecompose(F, R, output=system);
  Display(ccd, R);
```

$$ccd := [regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system]$$

$$\left[\left\{\begin{array}{l} y = 0 \\ x - 1 = 0 \end{array}\right., \left\{\begin{array}{l} x - 1 = 0 \\ y \neq 0 \end{array}\right., \left\{\begin{array}{l} y = 0 \\ x + 1 = 0 \end{array}\right., \left\{\begin{array}{l} x + 1 = 0 \\ y \neq 0 \end{array}\right., \left\{\begin{array}{l} y^2 + x^2 - 1 = 0 \\ x^2 - 1 \neq 0 \end{array}\right., \left\{\begin{array}{l} y^2 + x^2 - 1 \neq 0 \\ x^2 - 1 \neq 0 \end{array}\right.\right] \qquad \textbf{(8.1.3)}$$

A CCD is best described by a complex cylindrical tree (CCT).
Informally, a CCT **T** of $\mathbb{Q}[x_1,x_2,...,x_n]$ is a rooted tree with each non-root node described by a polynomial constraint $p(x_1,...,x_i)=0$, $p(x_1,...,x_i) \neq 0$, or "any x_i".
Moreover, the polynomial constraints in any path of **T** form a regular system such that
the union of their zero sets form a CCD.

The CCT can be displayed by the piecewise option.

```
> R := PolynomialRing([y, x]);
  F := [x^2+y^2-1];
  CylindricalDecompose(F, R, output=piecewise);
```

$$R := polynomial\_ring$$

$$F := [y^2 + x^2 - 1]$$

$$\begin{cases} \begin{cases} 1 & y = 0 \\ 1 & otherwise \end{cases} & x - 1 = 0 \\ \begin{cases} 1 & y = 0 \\ 1 & otherwise \end{cases} & x + 1 = 0 \\ \begin{cases} 1 & y^2 + x^2 - 1 = 0 \\ 1 & otherwise \end{cases} & otherwise \end{cases} \qquad \textbf{(8.1.4)}$$

If a set of polynomials, say **F**, is passed to CylindricalDecompose, then an F-invariant CCD
is computed. By F-invariant, we mean for any polynomial **f** of **F** and any cell **C** in the CCD,
either **C** is contained in the zero set of **f** or **C** has no intersection with the zero set of **f**.
One can use the command IsContained and Intersection to verify if the output satisfies the
invariance property.

```
> R := PolynomialRing([y, x]);
  f:= x^2+y^2-1; g := x*y-1;
  F := [f,g];
  lrs := CylindricalDecompose(F, R, output=system);
```

$$R := polynomial\_ring$$

$$f := y^2 + x^2 - 1$$

$$g := x\,y - 1$$

$$F := [y^2 + x^2 - 1, x\,y - 1]$$

$$lrs := [regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system, regular\_system] \qquad \textbf{(8.1.5)}$$

For example, the first complex cell, namely the zero set of the first regular system, is contained
in the zero set of **f**.

```
> rs1 := lrs[1];
  Display(rs1, R);
  csf := Triangularize([f], [1], R, output=lazard);
  cs1 := ConstructibleSet([rs1], R);
  IsContained(cs1, csf, R);
```

$$rs1 := regular\_system \begin{cases} y-1=0 \\ x=0 \end{cases}$$

$$csf := constructible\_set$$

$$cs1 := constructible\_set$$

$$true \qquad \textbf{(8.1.6)}$$

This cell has no intersection with the zero set of **g**.

```
> rs1 := lrs[1];
  Display(rs1, R);
  csg := Triangularize([g], [1], R, output=lazard);
  cs1 := ConstructibleSet([rs1], R);
  cs := Intersection(cs1, csg, R);
  IsEmpty(cs, R);
```

$$rs1 := regular\_system \begin{cases} y-1=0 \\ x=0 \end{cases}$$

$$csg := constructible\_set$$

$$cs1 := constructible\_set$$

$$cs := constructible\_set$$

$$true \qquad \textbf{(8.1.7)}$$

Besides a list of polynomials, a list of constraints (equations or inequations) are also allowed
to be the input of CylindricalDecompose. In this case, instead of a sign-invariant CCD, a (smaller) truth-invariant CCD will be computed.

```
> R := PolynomialRing([y, x]);
  F := [f=0,g=0];
  cad := CylindricalDecompose(F, R, output=piecewise);
```

$$R := polynomial\_ring$$

$$F := [y^2+x^2-1=0,\ yx-1=0]$$

$$cad := \begin{cases} \begin{cases} 1 & yx-1=0 \\ 1 & otherwise \end{cases} & x^4-x^2+1=0 \\ 1 & otherwise \end{cases} \qquad \textbf{(8.1.8)}$$

Several other output formats are supported, which are useful in some context.
If the input is a list of constraints, the options ''output=cct" and "output=tree" generate a partial CCT
expressing exactly their complex zeros while the options "output=ccd", "output= fulltree" and "output=piecewise" will generate a complete CCT.
The options "output=tree" and "output=fulltree" represent the CCT in a nested list style;

```
> R := PolynomialRing([y, x]);
  F := [f=0,g=0];
  cad := CylindricalDecompose(F, R, output=cct);
  Display(cad, R);
  cad := CylindricalDecompose(F, R, output='ccd');
  Display(cad, R);
  cad := CylindricalDecompose(F, R, output='piecewise');
  cad := CylindricalDecompose(F, R, output='tree');
  cad := CylindricalDecompose(F, R, output='fulltree');
```

$$R := polynomial\_ring$$

$$F := [y^2 + x^2 - 1 = 0, yx - 1 = 0]$$

$$cad := c\_c\_t$$

$$\left[\left\{\begin{array}{c} xy - 1 = 0 \\ x^4 - x^2 + 1 = 0 \\ x \neq 0 \end{array}\right\}\right]$$

$$cad := c\_c\_d$$

$$\left[\left\{\begin{array}{c} xy - 1 = 0 \\ x^4 - x^2 + 1 = 0 \\ x \neq 0 \end{array}\right\}, \left\{\begin{array}{c} x^4 - x^2 + 1 = 0 \\ xy - 1 \neq 0 \end{array}\right\}, x^4 - x^2 + 1 \neq 0\right]$$

$$cad := \left\{\begin{array}{cc} \left\{\begin{array}{cc} 1 & yx - 1 = 0 \\ 1 & otherwise \end{array}\right\} & x^4 - x^2 + 1 = 0 \\ 1 & otherwise \end{array}\right.$$

$$cad := [1, [x^4 - x^2 + 1 = 0, [yx - 1 = 0]]]$$

$$cad := [1, [x^4 - x^2 + 1, [yx - 1], [1]], [1, [1]]] \qquad \textbf{(8.1.9)}$$

8.2 Cylindrical algebraic decomposition of the real

space

The command CylindricalAlgebraicDecompose is used to compute a cylindrical algebraic decomposition of the real space. It supports different inputs, like list of polynomails, list of polynomial constraints, or list of list of polynomial constraints. It also provides different output formats, such as **'output'='piecewise', 'output'='tree', output='list', output='cadcell', output='rootof', output='cad'**. The default output option is **output='cad'**.

If the input **F** is a list of polynomials, an F-invariant CAD, which means that any polynomial of F is sign-invariant on any cell of the output CAD, is computed.

```
> restart; with(RegularChains): with(SemiAlgebraicSetTools):
  R := PolynomialRing([y, x]);
  F := [y^2-x, x-1];
  cad := CylindricalAlgebraicDecompose(F, R);
```

$$R := polynomial\_ring$$
$$F := [y^2 - x, x - 1]$$
$$cad := c\_a\_d \qquad (8.2.1)$$

By default, the output is a **'c_a_d'** type. One can use Display or Info to show its cells.

```
> Display(cad, R);
```

$$\left[\left\{\begin{array}{l} y = y \\ x < 0 \end{array}\right., \left\{\begin{array}{l} y < 0 \\ x = 0 \end{array}\right., \left\{\begin{array}{l} y = 0 \\ x = 0 \end{array}\right., \left\{\begin{array}{l} 0 < y \\ x = 0 \end{array}\right., \left\{\begin{array}{l} y < -\sqrt{x} \\ 0 < x \wedge x < 1 \end{array}\right., \qquad (8.2.2)$$
$$\left\{\begin{array}{l} y = -\sqrt{x} \\ 0 < x \wedge x < 1 \end{array}\right., \left\{\begin{array}{l} -\sqrt{x} < y \wedge y < \sqrt{x} \\ 0 < x \wedge x < 1 \end{array}\right., \left\{\begin{array}{l} y = \sqrt{x} \\ 0 < x \wedge x < 1 \end{array}\right.,$$
$$\left\{\begin{array}{l} \sqrt{x} < y \\ 0 < x \wedge x < 1 \end{array}\right., \left\{\begin{array}{l} y < -1 \\ x = 1 \end{array}\right., \left\{\begin{array}{l} y = -1 \\ x = 1 \end{array}\right.,$$
$$\left\{\begin{array}{l} -1 < y \wedge y < 1 \\ x = 1 \end{array}\right., \left\{\begin{array}{l} y = 1 \\ x = 1 \end{array}\right., \left\{\begin{array}{l} 1 < y \\ x = 1 \end{array}\right., \left\{\begin{array}{l} y < -\sqrt{x} \\ 1 < x \end{array}\right.,$$
$$\left\{\begin{array}{l} y = -\sqrt{x} \\ 1 < x \end{array}\right., \left\{\begin{array}{l} -\sqrt{x} < y \wedge y < \sqrt{x} \\ 1 < x \end{array}\right., \left\{\begin{array}{l} y = \sqrt{x} \\ 1 < x \end{array}\right., \left\{\begin{array}{l} \sqrt{x} < y \\ 1 < x \end{array}\right.\right]$$

```
> Info(cad, R);
```

$$[[x < 0, y = y], [x = 0, y < 0], [x = 0, y = 0], [x = 0, 0 < y], [0 < x \wedge x < 1, y < -\sqrt{x}], [0 < x \wedge x < 1, y = -\sqrt{x}], [0 < x \wedge x < 1, -\sqrt{x} < y \wedge y < \sqrt{x}], [0 < x \wedge x < 1, y = \sqrt{x}], [0 < x \wedge x < 1, \sqrt{x} < y], [x = 1, y < -1], [x = 1, \qquad (8.2.3)$$

$$y = -1], [x = 1, -1 < y \wedge y < 1], [x = 1, y = 1], [x = 1, 1 < y], [1 < x, y < -\sqrt{x}], [1 < x, y = -\sqrt{x}], [1 < x, -\sqrt{x} < y \wedge y < \sqrt{x}], [1 < x, y = \sqrt{x}], [1 < x, \sqrt{x} < y]]$$

The tree data structure of the CAD is best shown by the option 'output'= 'piecewise'.

```
> R := PolynomialRing([y, x]);
  F := [y^2-x];
  cad := CylindricalAlgebraicDecompose(F, R, output=piecewise);
```

$$R := polynomial\_ring$$

$$F := [y^2 - x]$$

$$cad := \begin{cases} 1 & x < 0 \\ \begin{cases} 1 & y < 0 \\ 1 & y = 0 \\ 1 & 0 < y \end{cases} & x = 0 \\ \begin{cases} 1 & y < -\sqrt{x} \\ 1 & y = -\sqrt{x} \\ 1 & -\sqrt{x} < y \wedge y < \sqrt{x} \\ 1 & y = \sqrt{x} \\ 1 & \sqrt{x} < y \end{cases} & 0 < x \end{cases} \qquad \textbf{(8.2.4)}$$

The input can be a list of polynomial constraints, which incodes a conjunction formula or a semi-algebraic system. The Display command shows all the cells of the output CAD.

```
> R := PolynomialRing([y, x]);
  F := [y^2-x=0, x-1=0];
  cad := CylindricalAlgebraicDecompose(F, R);
  Display(cad, R);
```

$$R := polynomial\_ring$$

$$F := [y^2 - x = 0, x - 1 = 0]$$

$$cad := c\_a\_d$$

(8.2.5)

$$\left[\left[\begin{cases} y = y \\ x < 1 \end{cases}, \begin{cases} y < -1 \\ x = 1 \end{cases}, \begin{cases} y = -1 \\ x = 1 \end{cases}, \begin{cases} -1 < y \wedge y < 1 \\ x = 1 \end{cases}, \begin{cases} y = 1 \\ x = 1 \end{cases}, \begin{cases} 1 < y \\ x = 1 \end{cases}, \begin{cases} y = y \\ 1 < x \end{cases}\right]\right] \quad \textbf{(8.2.5)}$$

To see all the true cells, that is the cells satisfying the input constraints, **'output'='cadcell'** can be used.

```
> R := PolynomialRing([y, x]);
  cad := CylindricalAlgebraicDecompose([x^2+y^2-1=0, x*y-1/2=0], R,
  output=cadcell);
  Display(cad, R);
```

$$R := polynomial\_ring$$
$$cad := [cad\_cell, cad\_cell]$$
$$\left[\left[\begin{cases} y = \frac{1}{2x} \\ x = -\frac{\sqrt{2}}{2} \end{cases}, \begin{cases} y = \frac{1}{2x} \\ x = \frac{\sqrt{2}}{2} \end{cases}\right]\right] \quad \textbf{(8.2.6)}$$

As can be seen, the option **'output'='cadcell'** does not attempt to make complete back substitution.
The option **'output'='rootof'** instead supports complete back substitution and tries to merge adjacent cells
to produce compact formula.

```
> R := PolynomialRing([y, x]):
  cad := CylindricalAlgebraicDecompose([x^2+y^2-1=0, x*y-1/2=0], R,
  output=rootof);
```

$$cad := \left[\left[x = -\frac{\sqrt{2}}{2}, y = -\frac{\sqrt{2}}{2}\right], \left[x = \frac{\sqrt{2}}{2}, y = \frac{\sqrt{2}}{2}\right]\right] \quad \textbf{(8.2.7)}$$

```
> R := PolynomialRing([y, x]):
  cad := CylindricalAlgebraicDecompose([x^2+y^2-1<=0], R, output=
  rootof);
```

$$cad := \left[\left[-1 \le x \wedge x \le 1, -\sqrt{-x^2+1} \le y \wedge y \le \sqrt{-x^2+1}\right]\right] \quad \textbf{(8.2.8)}$$

The input can also be a list of list of polynomial constraints, which represents a formula in disjunctive normal form or a union of semi-algebraic systems.

```
> R := PolynomialRing([y, x]);
  F := [[y^2-x=0], [x-1=0]];
  cad := CylindricalAlgebraicDecompose(F, R, output=cadcell);
  Display(cad, R);
```

$$R := polynomial\_ring$$

$$F := [[y^2 - x = 0], [x - 1 = 0]]$$

$$cad := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell]$$

$$\left[\begin{cases} y = 0 \\ x = 0 \end{cases}, \begin{cases} y = -\sqrt{x} \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} y = \sqrt{x} \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} y < -1 \\ x = 1 \end{cases}, \right. \tag{8.2.9}$$

$$\begin{cases} y = -1 \\ x = 1 \end{cases}, \begin{cases} -1 < y \wedge y < 1 \\ x = 1 \end{cases}, \begin{cases} y = 1 \\ x = 1 \end{cases}, \begin{cases} 1 < y \\ x = 1 \end{cases},$$

$$\left. \begin{cases} y = -\sqrt{x} \\ 1 < x \end{cases}, \begin{cases} y = \sqrt{x} \\ 1 < x \end{cases} \right]$$

Note that only cells making the input formula satisfied are showns. To see all the cells, one
can use the default option **'output'='cad'** or the option **'output'='allcell'**.

```
> R := PolynomialRing([y, x]);
  F := [[y^2-x=0], [x-1=0]];
  cad := CylindricalAlgebraicDecompose(F, R, output=allcell);
  Display(cad, R);
```

$$R := polynomial\_ring$$

$$F := [[y^2 - x = 0], [x - 1 = 0]]$$

$$cad := [cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell, cad\_cell]$$

$$\left[\begin{cases} y = y \\ x < 0 \end{cases}, \begin{cases} y < 0 \\ x = 0 \end{cases}, \begin{cases} y = 0 \\ x = 0 \end{cases}, \begin{cases} 0 < y \\ x = 0 \end{cases}, \right. \tag{8.2.10}$$

$$\begin{cases} y < -\sqrt{x} \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} y = -\sqrt{x} \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} -\sqrt{x} < y \wedge y < \sqrt{x} \\ 0 < x \wedge x < 1 \end{cases},$$

$$\begin{cases} y = \sqrt{x} \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} \sqrt{x} < y \\ 0 < x \wedge x < 1 \end{cases}, \begin{cases} y < -1 \\ x = 1 \end{cases},$$

$$\begin{cases} y = -1 \\ x = 1 \end{cases}, \begin{cases} -1 < y \wedge y < 1 \\ x = 1 \end{cases}, \begin{cases} y = 1 \\ x = 1 \end{cases}, \begin{cases} 1 < y \\ x = 1 \end{cases},$$

$$\left[\begin{cases} y < -\sqrt{x} \\ 1 < x \end{cases}, \begin{cases} y = -\sqrt{x} \\ 1 < x \end{cases}, \begin{cases} -\sqrt{x} < y \wedge y < \sqrt{x} \\ 1 < x \end{cases}, \begin{cases} y = \sqrt{x} \\ 1 < x \end{cases}, \begin{cases} \sqrt{x} < y \\ 1 < x \end{cases}\right]$$

Several other output formats are supported, including **'output'='tree', 'output'='list'**.

```
> R := PolynomialRing([y, x]);
  F := [y^2-x];
  cad := CylindricalAlgebraicDecompose(F, R, output=list);
```

$$R := polynomial\_ring$$
$$F := [y^2 - x]$$
$$cad := \left[\left[[1, 1], \left[regular\_chain, \left[\left[-\frac{1}{2}, -\frac{1}{2}\right], [0, 0]\right]\right]\right], \left[[2, 1], \left[regular\_chain, \left[[0, 0], \left[-\frac{1}{2}, -\frac{1}{2}\right]\right]\right]\right], [[2, 2], [regular\_chain, [[0, 0], [0, 0]]]], \left[[2, 3], \left[regular\_chain, \left[[0, 0], \left[\frac{1}{2}, \frac{1}{2}\right]\right]\right]\right], \left[[3, 1], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[-\frac{3}{2}, -\frac{3}{2}\right]\right]\right]\right], \left[[3, 2], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[-\frac{1482911}{2097152}, -\frac{741455}{1048576}\right]\right]\right]\right], \left[[3, 3], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], [0, 0]\right]\right]\right], \left[[3, 4], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[\frac{741455}{1048576}, \frac{46341}{65536}\right]\right]\right]\right], \left[[3, 5], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[\frac{3}{2}, \frac{3}{2}\right]\right]\right]\right]\right] \quad \textbf{(8.2.11)}$$

```
> R := PolynomialRing([y, x]);
  F := [y^2-x];
  cad := CylindricalAlgebraicDecompose(F, R, output=tree);
```

$$R := polynomial\_ring$$
$$F := [y^2 - x]$$
$$cad := \left[1, \left[\left[[-1, x, 1], \left[regular\_chain, \left[\left[-\frac{1}{2}, -\frac{1}{2}\right]\right]\right]\right], \left[\left[[1], \left[regular\_chain, \left[\left[-\frac{1}{2}, -\frac{1}{2}\right], [0, 0]\right]\right]\right]\right], \left[[[x, 1], [regular\_chain, [[0, 0]]]], \left[\left[[-1, y, 1], \left[regular\_chain, \left[[0, 0], \left[-\frac{1}{2}, -\frac{1}{2}\right]\right]\right]\right]\right], [[[y, 1], \right.\right.\right. \quad \textbf{(8.2.12)}$$

$$[regular\_chain, [[0, 0], [0, 0]]]], \left[\left[[1, y, 1], \left[regular\_chain, \left[[0, 0], \left[\frac{1}{2}, \frac{1}{2}\right]\right]\right]\right]\right], \left[\left[[1, x, 1], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right]\right]\right]\right], \left[\left[[-1, -y^2 + x, 1], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[-\frac{3}{2}, -\frac{3}{2}\right]\right]\right]\right], \left[\left[[-y^2 + x, 1], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[-\frac{1482911}{2097152}, -\frac{741455}{1048576}\right]\right]\right]\right], \left[\left[[-y^2 + x, 1, -y^2 + x, 2], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], [0, 0]\right]\right]\right], \left[\left[[-y^2 + x, 2], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[\frac{741455}{1048576}, \frac{46341}{65536}\right]\right]\right]\right], \left[\left[[1, -y^2 + x, 2], \left[regular\_chain, \left[\left[\frac{1}{2}, \frac{1}{2}\right], \left[\frac{3}{2}, \frac{3}{2}\right]\right]\right]\right]\right]$$

The default algorithm of CylindricalAlgebraicDecompose is an incremental algorithm (CM) based on triangular decompositions proposed by Changbo Chen and Marc Moreno Maza on ASCM' 2012. An alternative algorithm (CMXY) proposed by Changbo Chen, Marc Moreno Maza, Bican Xia and Lu Yang on ISSAC' 2009, which is also the first generation of algorithm for computing CAD based on triangular decompositions, is available by the option **'method'= 'recursive'**.

```
> R := PolynomialRing([y, x]);
  F := [y^2-x];
  cad := CylindricalAlgebraicDecompose(F, R, method=recursive,
  output=piecewise);
```

$$R := polynomial\_ring$$

$$F := [y^2 - x]$$

$$cad := \begin{cases} \begin{cases} 1 & y < -\sqrt{x} \\ 1 & y = -\sqrt{x} \\ 1 & -\sqrt{x} < y \wedge y < \sqrt{x} \\ 1 & y = \sqrt{x} \\ 1 & \sqrt{x} < y \end{cases} & 0 < x \\ \begin{cases} 1 & y < 0 \\ 1 & y = 0 \\ 1 & 0 < y \end{cases} & x = 0 \\ 1 & x < 0 \end{cases} \tag{8.2.13}$$

The CM algorithm can take advantage of equational constraints in a single semi-algebraic system.
That is, if the input is a list of polynomial constraints involving equations, by default, the option
'optimization'='EC' is enabled. To disable such optimization, one sets **'optimization'='false'**.

```
> R := PolynomialRing([y, x]);
  F := [x^2+y^2-1=0, y^2-x=0];
  cad := CylindricalAlgebraicDecompose(F, R, optimization='EC',
  output=allcell):
  nops(cad);

  cad2 := CylindricalAlgebraicDecompose(F, R, optimization='false',
  output=allcell):
  nops(cad2);
```

$$R := polynomial\_ring$$

$$F := [y^2 + x^2 - 1 = 0, y^2 - x = 0]$$

$$9$$

$$5\,3 \qquad \textbf{(8.2.14)}$$

If input is a list of list of polynomial constraints, by default an algorithm (RC-TTICAD) for computing truth-table invariant CAD is used to compute a smaller CAD. One could use the option **'optimization'='TTICAD'** or **'optimization'='EC'** to enable or disable it.

```
> R := PolynomialRing([y, x]);
  F := [[(x-2)^2+(y-4)^2=0], [x^2+y^2-1=0, y^2-x=0]];
  cad := CylindricalAlgebraicDecompose(F, R, optimization='TTICAD',
  output=allcell):
  nops(cad);

  cad2 := CylindricalAlgebraicDecompose(F, R, optimization='EC',
  output=allcell):
  nops(cad2);
```

$$R := polynomial\_ring$$

$$F := [[(x-2)^2 + (y-4)^2 = 0], [y^2 + x^2 - 1 = 0, y^2 - x = 0]]$$

$$3\,3$$

$$6\,5 \qquad \textbf{(8.2.15)}$$

To get a sample point of a CAD cell, the function SamplePoints can be called. Here no cost occurs since sample points are computed along the computation of the CAD and are stored in the type cad cell. A sample point is encoded by the type box, which is represented by a regular chain and an isolation cube. Such a representation allows one to easily test if the sign of a polynomial at the sample

point by calling the function SignAtBox.

```
> R := PolynomialRing([y, x]):
  cad := CylindricalAlgebraicDecompose([x^2+y^2-1=0, x*y-1/2=0],R,
  output=cadcell);
  Display(cad, R);
  sp := map(SamplePoints, cad, R);
  Display(sp, R);
  s := SignAtBox(x^2+y^2-2, sp[1], R);
```

$$cad := [cad\_cell, cad\_cell]$$

$$\left[\left\{ y = \frac{1}{2x},\ x = -\frac{\sqrt{2}}{2} \right\},\ \left\{ y = \frac{1}{2x},\ x = \frac{\sqrt{2}}{2} \right\}\right]$$

$$sp := [box, box]$$

$$\left[\left\{ y = \left[-\frac{46341}{65536}, -\frac{1482907}{2097152}\right],\ x = \left[-\frac{46341}{65536}, -\frac{741455}{1048576}\right] \right\},\ \left\{ y = \left[\frac{185363}{262144}, \frac{741457}{1048576}\right],\ x = \left[\frac{741455}{1048576}, \frac{46341}{65536}\right] \right\}\right]$$

$$s := -1 \qquad \textbf{(8.2.16)}$$

8.3 Quantifier elimination

As one of the main applications of cylindrical algebraic decomposition, quantifier elimination is fully supported in the library. The function for doing quantifier elimination is QuantifierElimination.
The user interface of Quantifier Elimination replies one some features of the **Logic** library of Maple.
We introduce in addition the existential quantifier "&E" and the universal quantifier "&A".

Suppose we'd like to solve the following QE problem (due to Davenport and Heintz):

$(\exists\, c)\ (\forall\, b, a)\ ((a = d \wedge b = c) \vee (a=c \wedge b = 1)) \Rightarrow a^2 = b.$

```
> f := &E([c]), &A([b, a]), ((a=d) &and (b=c)) &or ((a=c) &and (b=
```

```
  1)) &implies (a^2=b);
```

$$f := \&E([c]), \&A([b, a]), (((a = d)\ \&and\ (b = c))\ \&or\ ((a = c)\ \&and\ (b = 1)))\ \&implies\ (a^2 = b) \qquad \textbf{(8.3.1)}$$

```
> out := QuantifierElimination(f);
```

$$out := (d - 1 = 0)\ \&or\ (d + 1 = 0) \qquad \textbf{(8.3.2)}$$

Note that in the previous example, no variable order is specified.
In such case, the function will try to find the best elimination order according to some heuristic strategy.

```
> R := PolynomialRing([x, a, b, c]);
  f := &E([x]), a*x^2+b*x+c=0;
  out := QuantifierElimination(f, R);
```

$$R := polynomial\_ring$$

$$f := \&E([x]), a\,x^2 + b\,x + c = 0$$

$$out := \&or(c = 0, \&and(c < 0, b < 0, a = 0), \&and(c < 0, b < 0, 0 < a), \&and(c < 0, b < 0, 4\,a\,c - b^2 = 0), \&and(c < 0, b < 0, a < 0, 4\,a\,c - b^2 < 0), \&and(c < 0, b = 0, 0 < a), \&and(c < 0, 0 < b, a = 0), \&and(c < 0, 0 < b, 0 < a), \&and(c < 0, 0 < b, 4\,a\,c - b^2 = 0), \&and(c < 0, 0 < b, a < 0, 4\,a\,c - b^2 < 0), \&and(0 < c, b < 0, a < 0), \&and(0 < c, b < 0, a = 0), \&and(0 < c, b < 0, 4\,a\,c - b^2 = 0), \&and(0 < c, b < 0, 0 < a, 4\,a\,c - b^2 < 0), \&and(0 < c, b = 0, a < 0), \&and(0 < c, 0 < b, a < 0), \&and(0 < c, 0 < b, a = 0), \&and(0 < c, 0 < b, 4\,a\,c - b^2 = 0), \&and(0 < c, 0 < b, 0 < a, 4\,a\,c - b^2 < 0)) \qquad \textbf{(8.3.3)}$$

By default, QuantifierElimination returns the standard quantifier free formula, namely Tarski formula.
Extended Tarski formula is supported by the option 'output'='rootof'.

```
> f := &E([x]), a*x^2+b*x+c=0;
  out := QuantifierElimination(f,'output'='rootof');
```

$$f := \&E([x]), x^2\,a + x\,b + c = 0$$

$$out := \&or\left((a < 0)\ \&and\ \left(\frac{b^2}{4\,a} \le c\right), (a = 0)\ \&and\ (b \ne 0), \&and(a = 0, b = 0, c = 0), (0 < a)\ \&and\ \left(c \le \frac{b^2}{4\,a}\right)\right) \qquad \textbf{(8.3.4)}$$

More examples on generating extended Tarski Formula.

```
> f := &E([y]), y^2+x^2=2;
  out := QuantifierElimination(f, output=rootof);
```

$$f := \&E([y]),\ x^2 + y^2 = 2$$

$$out := \left(-\sqrt{2} \le x\right) \&and \left(x \le \sqrt{2}\right) \qquad \textbf{(8.3.5)}$$

```
> f := &E([y]), y^4+x^4=2;
  out := QuantifierElimination(f, output=rootof);
```

$$f := \&E([y]),\ x^4 + y^4 = 2$$

$$out := \left(RootOf\left(\_Z^4 - 2, index = real_1\right) \le x\right) \&and \left(x \le RootOf\left(\_Z^4 - 2, index = real_2\right)\right) \qquad \textbf{(8.3.6)}$$

```
> evalf(op(1, out)); evalf(op(2, out));
```

$$-1.189207115 \le x$$

$$x \le 1.189207115 \qquad \textbf{(8.3.7)}$$

An application for computing control Lyapunov function.

```
> f1 := -x_1+u: f2 := -x_1-x_2^3:
  V := a_1*x_1^2+a_2*x_2^2;
  Vt := diff(V, x_1)*f1 + diff(V, x_2)*f2;
```

$$V := a\_1\, x\_1^2 + a\_2\, x\_2^2$$

$$Vt := 2\, a\_1\, x\_1\, (-x\_1 + u) + 2\, a\_2\, x\_2\, \left(-x\_2^3 - x\_1\right) \qquad \textbf{(8.3.8)}$$

```
> QuantifierElimination( &A([x_1,x_2]), &E([u]), (x_1<>0) &or
  (x_2<>0) &implies ((V>0) &and (Vt<0)) );
```

$$(0 < a\_2) \&and (0 < a\_1) \qquad \textbf{(8.3.9)}$$

```
> QuantifierElimination( &A([x_1, x_2]), &E([u]), (u=b_1*x_1+b_2*
  x_2) &and (a_1>0) &and (a_2>0) &and ((x_1<>0) &or (x_2<>0)
  &implies ((Vt<0))) );
```

$$\&and(0 < a\_2, 0 < a\_1, b\_2\, a\_1 - a\_2 = 0, b\_1 < 1) \qquad \textbf{(8.3.10)}$$

Simplification of the output.

Without simplification:

```
> ff := &E([i,j]), (0 <= i) &and (i <= n) &and (0 <= j) &and (j <=
  n) &and (t = n - j) &and (p = i + j);

  R := PolynomialRing([i,j,p,t,n]);

  sols := QuantifierElimination(ff, R, output=rootof,
```

```
  simplification=false);
```

$$ff := \&E([i,j]), ((((((0 \le i) \,\&and\, (i \le n)) \,\&and\, (0 \le j)) \,\&and\, (j \le n)) \,\&and\, (t = n - j)) \,\&and\, (p = i + j)$$

$$R := polynomial\_ring$$

$$sols := \&or(\&and(n = 0, t = n, p = 0), \&and(0 < n, t = 0, p = n), \&and(0 < n, t = 0, n < p, p < 2\,n), \&and(0 < n, t = 0, p = 2\,n), \&and(0 < n, 0 < t, t < n, p = -t + n), \&and(0 < n, 0 < t, t < n, -t + n < p, p < -t + 2\,n), \&and(0 < n, 0 < t, t < n, p = -t + 2\,n), \&and(0 < n, t = n, p = 0), \&and(0 < n, t = n, 0 < p, p < n), \&and(0 < n, t = n, p = n)) \quad \textbf{(8.3.11)}$$

With simplification:

```
> sols := QuantifierElimination(ff, R, output=rootof,
  simplification=L4);
```

$$sols := \&and(0 \le n, 0 \le t, t \le n, n - t \le p, p \le -t + 2\,n) \quad \textbf{(8.3.12)}$$

9. Computing Limits: from rational functions to topological closures

The AlgebraicGeometryTools provides facilities for studying algebraic curves, surfaces and algebraic sets of higher dimension. The commands currently available mainly focus on computing limits of "family of objects" like limits of family of secants in the case of tangent cone computation.

Different flavors of limit points

The command LimitPoints is part of AlgebraicGeometryTools package. The purpose of this command is to compute the limit points of a constructible set *rc* of dimension one. These limit points are corresponding to the values of the free variable of regular chain rc vanishing the product of the initials of the polynomials of rc.The command LimitPoints comes in two different types: 1) limit points w.r.t Zariski topology, 2) limit points w.r.t Euclidean topology. If one wants to compute the limit points of a constructible set with respect to Zariski topology, then one needs to use LimitPoints as following:

```
> restart:with(RegularChains): with(ChainTools):
  with(AlgebraicGeometryTools):
```

```
R := PolynomialRing([x, y, z]):
rc := Chain([y^5-z^4, x*z-y^2], Empty(R), R):
lm := LimitPoints(rc, R):
Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x = 0 \\ y = 0 \\ z = 0 \end{array}\right\}\right] \qquad (9.1.1)$$

Furthermore, LimitPoints command uses a variable called coefficient in order to indicate what the coefficient ring for the computations is. For the current implementation coefficient can accept only complex and real corresponding to the limit points w.r.t Zariski and Euclidean topology, respectively. The variable coefficient is set to complex by default. Therfore the following is equvalent to the computations above for *lm*.

```
> rc := Chain([y^5-z^4, x*z-y^2], Empty(R), R):
  lm := LimitPoints(rc, R, coefficient = complex):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x = 0 \\ y = 0 \\ z = 0 \end{array}\right\}\right] \qquad (9.1.2)$$

When the coefficient is set as complex, then the output of LimitPoints is called the complex limit points or for short only limit points of the regular chain *rc*. When the coefficient is set as real then this output is called the real limit points of the regular chain *rc*. The following shows how to compute the real limit points of the regular chain *rc*.

```
> rc := Chain([y^5-z^4, x*z-y^2], Empty(R), R):
  lm := LimitPoints(rc, R, coefficient = real):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x = 0 \\ y = 0 \\ z = 0 \end{array}\right\}\right] \qquad (9.1.3)$$

The command LimitPoints also have another option in order to indicate for which value of the free variable of regular chain rc, one want to compute the limit points of rc. In order to indicate this option, one new argument should be passed to LimitPoints which is a list of a single univariate polynomial in free variable of rc whose zeros are the corresponding values one want to compute the limit points of

rc at.

The following line shows how LimitPoints returns all the non-trivial limit points of rc.

```
> rc:= Chain([y^5-z^4*(z+1)^5,x*z*(z+1)^2-y^2],Empty(R), R):
  lm := LimitPoints(rc, R); Display(lm, R);
```

$$lm := [regular\_chain, regular\_chain, regular\_chain]$$

$$\left[\left\{\begin{array}{l} x=0 \\ y=0 \\ z=0 \end{array}\right., \left\{\begin{array}{l} x+1=0 \\ y=0 \\ z+1=0 \end{array}\right., \left\{\begin{array}{l} x^4-x^3+x^2-x+1=0 \\ y=0 \\ z+1=0 \end{array}\right.\right] \qquad \textbf{(9.1.4)}$$

While the following computes the limit points only relatd to some values of the free variable of the regular chain rc.

```
> lm := LimitPoints(rc, R, [z]):Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x=0 \\ y=0 \\ z=0 \end{array}\right.\right] \qquad \textbf{(9.1.5)}$$

Real limits vs complex limits

While for some cases real limit points might be equal to the complex limit points, this is not the case all the time.
The following example shows an example for which the real limit points are different from the complex ones, even if the complx limit points all have rational coefficients.

```
> R := PolynomialRing([y, x, z]):
  rc := Chain([z^5+x^4-2*x^3+x^2, y*z^4+x^3-x^2], Empty(R), R):
  lm := LimitPoints(rc, R, coefficient = complex):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} y=0 \\ x=0 \\ z=0 \end{array}\right.\right] \qquad \textbf{(9.2.1)}$$

```
> m := LimitPoints(rc, R, coefficient = real):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} y=0 \\ x=0 \\ z=0 \end{array}\right\}\right] \qquad (9.2.2)$$

For the following example the real limit points and complex ones are all equal.

```
> R := PolynomialRing([x, y, z]):
  rc:=Chain([y^(3)-2* y^(3)+y^(2)+z^(5), z^(4)* x+y^(3)-y^(2)],
  Empty(R),R):
  lm := LimitPoints(rc, R, coefficient = complex):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x=0 \\ y=0 \\ z=0 \end{array}\right\}, \left\{\begin{array}{l} x=0 \\ y-1=0 \\ z=0 \end{array}\right\}\right] \qquad (9.2.3)$$

```
> lm := LimitPoints(rc, R, coefficient = real):
  Display(lm, R);
```

$$\left[\left\{\begin{array}{l} x=0 \\ y-1=0 \\ z=0 \end{array}\right\}\right] \qquad (9.2.4)$$

Different output formats of LimitPoints

The command LimitPoints have two different output formats in which the variable ouput is responsible to control this option. A user can indicate output variable to be set as chain, which is the default of LimitPoints command, or rootof, each of which gives a different representation of the limit points of rc.

```
> R := PolynomialRing([x, y, z]):
  rc := Chain([y^4-(z^2-2)^4, (z^2-2)*x-y^2], Empty(R), R):
  lm := LimitPoints(rc, R, [z^2-2], output = rootof, coefficient =
  real):
  Display(lm, R);
```

$$[[z = RootOf(\_Z^2 - 2), y = 0, x = 0]] \qquad (9.3.1)$$

```
> lm := LimitPoints(rc, R, [z^2-2], output = chain, coefficient =
  real):
  Display(lm, R);
```

(9.3.2)

$$\left[\left\{\begin{array}{c} x=0 \\ y=0 \\ z^2-2=0 \end{array}\right\}\right] \tag{9.3.2}$$

Computing the branches of the one-dimensional constructible sets

The command RegularChainBranches is part of AlgebraicGeometryTools package. The goal of this command is to compute a parametrization of the branches of a constructible set of dimension one represented by a regular chain. These parametrizations are actually the Puiseux parametrization of a constructible set when the free variable approaches zero. The following example show how to use this command without any extra option:

```
> with(AlgebraicGeometryTools):
  R := PolynomialRing([x, y, z]):
  rc := Chain([-z^2+y, x*z-y^2], Empty(R), R):
  br := RegularChainBranches(rc, R, [z]);
```

$$br := [[z=T, y=T^2, x=T^3]] \tag{9.4.1}$$

```
> rc := Chain([y^2*z+y+1, (z+2)*z*x^2+(y+1)*(x+1)], Empty(R), R):
  RegularChainBranches(rc, R, [z]);
  RegularChainBranches(rc, R, [z+2]);
```

$$\left[\left[z=T, y=-T-1, x=-\frac{1}{432}T^5+\frac{11}{432}T^4+\frac{5}{432}T^3-\frac{5}{216}T^2+\frac{1}{12}T-\frac{1}{2}\right], \left[z=T, y=-T-1, x=\frac{(T-2)\left(T^2+4\right)\left(T^2-9T-54\right)}{432}\right]\right]$$

$$\left[\left[z=T-2, y=\frac{1}{216}T^4+\frac{11}{216}T^3-\frac{5}{216}T^2-\frac{1}{12}T-\frac{1}{2}, x=\frac{9}{8}T^3+2T^2+4T-1\right], \left[z=T-2, y=-\frac{1}{216}T^4-\frac{11}{216}T^3+\frac{4}{27}T^2+\frac{1}{3}T+1, x=\frac{3}{8}T^3+\frac{1}{2}T^2+T-1\right]\right] \tag{9.4.2}$$

Like the LimitPoints command, RegularChainBranches also works in two different modes: 1) complex, and 2) real. To switch between two different modes, user can use coefficient variable which by default is set to be complex. The variable coefficient indicates in which coefficient ring, the coresponding parametrization of the branches will be computed. If it is set as real then it means that all the coefficients of the parametrizations are real numbers, complex, otherwise.

```
> rc:=Chain([y^(3)-2* y^(3)+y^(2)+z^(5), z^(4)* x+y^(3)-y^(2)],
  Empty(R),R):
  br := RegularChainBranches(rc, R,  [z], coefficient = complex);
```

$$br := \left[\left[z = T^2, y = -\frac{T^5\left(T^5 - 2\,RootOf(\_Z^2+1)\right)}{2}, x = \frac{T^2\left(T^{20} - 6\,T^{15}\,RootOf(\_Z^2+1) - 10\,T^{10} - 8\right)}{8}\right], \left[z = T^2, y = -\frac{T^5\left(T^5 + 2\,RootOf(\_Z^2+1)\right)}{2}, x = \frac{T^2\left(T^{20} + 6\,T^{15}\,RootOf(\_Z^2+1) - 10\,T^{10} - 8\right)}{8}\right], \left[z = T, y = T^5+1, x = -T^{11} - 2\,T^6 - T\right]\right] \quad (9.4.3)$$

```
> br := RegularChainBranches(rc, R, [z], coefficient = real);
```

$$br := [[z = T, y = T^5 + 1, x = -T^{11} - 2\,T^6 - T]] \quad (9.4.4)$$

In the parametrization of the branches of the constructible set represented as a regular chain, a new (parametric) variable is introduced. This variable by default is chosen to be T, but also can be cottroled by user. The following examples shows how this can be done:

```
> rc := Chain([-z^2+y, x*z-y^2], Empty(R), R):
  br := RegularChainBranches(rc, R, [z^2+1], coefficient = complex)
  ;
```

$$br := [[z = T + RootOf(\_Z^2+1), y = 2\,T\,RootOf(\_Z^2+1) + T^2 - 1, x = (-T^8 + 3\,T^6 + 3\,T^2 - 1)\,RootOf(\_Z^2+1) + 3\,T^7 - T^5 + T^3 - 3\,T]] \quad (9.4.5)$$

```
> rc := Chain([-z^2+y, x*z-y^2], Empty(R), R):
  br := RegularChainBranches(rc, R, [z^2+1], coefficient = complex,
  W);
```

$$br := [[z = W + RootOf(\_Z^2+1), y = 2\,W\,RootOf(\_Z^2+1) + W^2 - 1, x = (-W^8 + 3\,W^6 + 3\,W^2 - 1)\,RootOf(\_Z^2+1) + 3\,W^7 - W^5 + W^3 - 3\,W]] \quad (9.4.6)$$

Also it is possible to determine how many terms can be computed in the parametrization of the branches of the regular chain rc. This feature can be used as follow:
First create a list of accuracies whose elements is equal to the number of polynomials in rc.

```
> L:= [1,8];
```

$$L := [1, 8] \quad (9.4.7)$$

Then, we have:

```
> R := PolynomialRing([x, y, z]):
  rc := Chain([y^2*z+y+1, (z+2)*z*x^2+(y+1)*(x+1)], Empty(R), R):
  RegularChainBranches(rc, R, [z],L);
```

(9.4.8)

$$\left[\left[z = T,\ y = -T - 1,\ x = -\frac{1}{4897760256}\left(\left(T^4 - 2T^3 + 4T^2 - 8T + 16\right)\left(143T^8 - 396T^7 + 1134T^6 - 3402T^5 + 10935T^4 - 39366T^3 + 177147T^2 - 1594323T - 4782969\right)\left(T^4 + 2T^3 + 4T^2 + 8T + 16\right)(T - 2)\right)\right], \left[z = T,\ y = -T - 1,\ x = \frac{1}{4897760256}\left(\left(143T^8 - 396T^7 + 1134T^6 - 3402T^5 + 10935T^4 - 39366T^3 + 177147T^2 - 1594323T - 9565938\right)\left(T^4 - 2T^3 + 4T^2 - 8T + 16\right)\left(T^4 + 2T^3 + 4T^2 + 8T + 16\right)(T - 2)\right)\right]\right] \quad \textbf{(9.4.8)}$$

If now, we change the accrucaies, we have the following:

```
> L:= [1,4]:
  RegularChainBranches(rc, R, [z],L);
```

$$\left[\left[z = T,\ y = -T - 1,\ x = \frac{5}{139968}T^9 - \frac{7}{34992}T^8 + \frac{137}{139968}T^7 - \frac{1003}{139968}T^6 - \frac{37}{2187}T^5 + \frac{74}{2187}T^4 - \frac{17}{243}T^3 + \frac{4}{27}T^2 - \frac{1}{3}T + 1\right], \left[z = T,\ y = -T - 1,\ x = -\frac{5}{139968}T^9 + \frac{7}{34992}T^8 - \frac{137}{139968}T^7 + \frac{1003}{139968}T^6 + \frac{181}{139968}T^5 - \frac{181}{69984}T^4 + \frac{29}{3888}T^3 - \frac{5}{216}T^2 + \frac{1}{12}T - \frac{1}{2}\right]\right] \quad \textbf{(9.4.9)}$$

It is worth mentioning that when the list L is not presented, it means a default value is computed for the list L. This value is called list of accrucaies and it meets some requirements. These requirements can be found at Computing the Limit Points of the Quasi-component of a Regular Chain in Dimension One.

Computing limits of fractional multivariate polynomials (when origin is the singular point of denominator)

RationalFunctionLimit is a command of AlgebraicGeometryTools in order to compute the limit of the fraction of multivariate polynomials at a point where this point is an isolated zero of the denominator. RationalFunctionLimit accepts two arguments: 1) the fraction of multivariate polynomials q, 2) a list of all the variables equal to some values, indicating the point we aim at computing the limit of q at. The following examples demonstrate how this command works:

```
> restart: with(AlgebraicGeometryTools):
  RationalFunctionLimit(x^2*y*z^2/(x^4+y^4+z^4), [x = 0, y = 0, z =
  0]);
```

$$0 \quad \textbf{(9.5.1)}$$

```
> RationalFunctionLimit((w*z+x^2+y^2)/(w^2+x^2+y^2+z^2), [x = 0, y
  = 0, z = 0, w = 0]);
```

"no_finite_limit" **(9.5.2)**

```
> RationalFunctionLimit(x^6/(w^6+l^2+t^2+x^2+y^2+z^2), [x = 0, y =
  0, z = 0, w = 0, t = 0, l = 0]);
```

0 **(9.5.3)**

```
> RationalFunctionLimit(x*y*z*w/(w^4+x^4+y^4+z^4), [x = 0, y = 0, z
  = 0, w = 0]);
```

"no_finite_limit" **(9.5.4)**

```
> RationalFunctionLimit((x*y+x*z-y*z)/(x^2+y^2+z^2), [x = 0, y = 0,
  z = 0]);
```

"no_finite_limit" **(9.5.5)**

Computing tangent cone of algebraic curves

The command TangentCone is integrated into AlgebraicGeometryTools package. The goal of this command is to compute the tangent cone of a polynomial system of dimension one at some points on the curve. These points can do not necessarily have rational coefficients. The following shows how to use this command.

First introduce the polynomial ring:

```
> R := PolynomialRing([x, y, z]):
```

Second, introduce the points using a zero-dimensional regular chain:

```
> rc := Chain([z-1, y, x], Empty(R), R):
```

Then, introduce the polynomial system of dimension one, compute the tangent cone of this system at the point represented by rc:

```
> Gs := [x^2+y^2+z^2-1, x^2-y^2-z*(z-1)]:
  tc := TangentCone(rc, Gs, R);
```

$$tc := \{[[\_z - 1, 3\,\_x^2 - \_y^2], regular\_chain]\} \quad \textbf{(9.6.1)}$$

There are two different representations for the output of TangentCone; Since the tangent cone of the polynomial systems of dimension one are lines, therefore, those lines can be represented both with their euqations or slopes. The default is set as equations. Each format of the output can be used as follows:

```
> tc := TangentCone(rc, Gs, R, equations);
```

$$tc := \{[[\_z - 1, 3\,\_x^2 - \_y^2], regular\_chain]\} \quad \textbf{(9.6.2)}$$

```
> tc := TangentCone(rc, Gs, R, slopes);
```

$$tc := \{[[z, y - 1, 3\,x^2 - 1], regular\_chain], [[z, y^2 - 3, x - 1], regular\_chain]\} \quad \textbf{(9.6.3)}$$

In order to indicate the equations or the slopes of the tangent cone some new variables are used in order to distinguish the coordinate system of the equations and slopes from the original coordinate system. The default variables used for equation mode are the original names of variables where underscore is added as prefix to the variables. For slopes precentage added as prefix to original variables is used as default variables. However, the default for these variables can be controled by user as follows:

```
> tc := TangentCone(rc, Gs, R, slopes, [t, s, w]);
```

$$tc := \{[[w, s-1, 3t^2-1], regular\_chain], [[w, s^2-3, t-1], regular\_chain]\} \quad \textbf{(9.6.4)}$$

Computing the tangent plane of the multivariate polynomials

TangentPlane is another command of AlgebraicGeometryTools package of Maple Its goal is to compute the tangent plane of a algebraic surface represented by a polynomial at some point on the surface.

```
> rc := Chain([z, y-1, x], Empty(R), R):
  tp := TangentPlane(rc, x*y, R);
```

$$tp := [[\_x, rc]] \quad \textbf{(9.7.1)}$$

Computing the intersection multiplicity of a polynomial system of dimension zero

The command TriangularizeWithMultiplicity is part of AlgebraicGeometryTools package. This command targets the intersection multiplicity computations of a polynomial system of dimension zero.
This command first attempts to solve this system using Triangularize command of RegularChains Library without taking multiplicity into considerations. Then for each point represented by the zero-dimansional regular chains in the triangular decomposition of the input system, it computes the intersection multiplicity of the original system at this related point.

```
> R := PolynomialRing([z, y, x]):
  F := [x^2+y+z-1, y^2+x+z-1, z^2+x+y-1]:
  dec := TriangularizeWithMultiplicity(F, R):
  Display(dec, R);
```

$$\left[\left[1, \left\{\begin{array}{c} z-x=0 \\ y-x=0 \\ x^2+2x-1=0 \end{array}\right\}\right], \left[2, \left\{\begin{array}{c} z=0 \\ y=0 \\ x-1=0 \end{array}\right\}\right], \left[2, \left\{\begin{array}{c} z=0 \\ y-1=0 \\ x=0 \end{array}\right\}\right], \left[2, \right. \right. \quad \textbf{(9.8.1)}$$

$$\left[\begin{cases} z-1=0 \\ y=0 \\ x=0 \end{cases}\right]$$

Return to Index for Example Worksheets