The `RegularChains` library in MAPLE 10

F. Lemaire⋆, M. Moreno Maza†, Y. Xie†

⋆: LIFL, Université Lille 1, 59655 Villeneuve d'Ascq, France.

†: ORCCA, University of Western Ontario, London, Ontario, Canada.

{lemaire}@lifl.fr {moreno,yxie}@orcca.on.ca

Abstract

The `RegularChains` library provides facilities for symbolic computations with systems of polynomial equations. In particular, it allows to compute modulo a set of algebraic relations. Automatic case discussion (and recombination) handles zero-divisors and parameters. This permits triangular decomposition of polynomial equations.

1 Introduction

Performing calculations modulo a set of relations is a basic technique in algebra. For instance, computing the inverse of an integer modulo a prime integer or computing the inverse of the complex number $3+2\imath$ modulo the relation $\imath^2+1=0$. Computing modulo a set F containing more than one relation requires from F to have some mathematical structure. For instance, computing the inverse of $p = x + y$ modulo $F_0 = \{x^2 + y + 1, y^2 + x + 1\}$ is difficult unless one realizes that this question is equivalent to computing the inverse of p modulo $C = \{x^4+2x^2+x+2, y+x^2+1\}$. Indeed, from there one can simplify p using $y = -x^2-1$ leading to $q = -x^2+x-1$ and compute the inverse of q modulo x^4+2x^2+x+2 (using the extended Euclidean algorithm) leading to $-\frac{1}{2}x^3 - \frac{1}{2}x$. One commonly used mathematical structure for a set of algebraic relations is that of a *Gröbner basis* [2]. It is particularly well suited for deciding whether a quantity is null or not modulo a set of relations. For inverse computations, the notion of a *regular chain* is more adequate. For instance, computing the inverse of $p = x+y$ modulo the set $C = \{y^2-2x+1, x^2-3x+2\}$, which is both a Gröbner basis and a regular chain for the variable order of $y > x$, is easily answered in this latter point of view. Indeed, it naturally leads to consider the GCD of p and $C_y = y^2 - 2x + 1$ modulo the relation $C_x = x^2 - 3x + 2 = 0$, which is

$$\mathrm{GCD}(p, C_y, \{C_x\}) = \begin{cases} p & \text{if} \quad x = 1 \\ 1 & \text{if} \quad x = 2 \end{cases}$$

This shows that p has no inverse if $x = 1$ and has an inverse (which can be computed and which is $-y+2$) if $x = 2$.

The notion of a regular chain was introduced by *Kalkbrener* in [10] and extended that of a *triangular set* (as defined in [11]). Kalkbrener pointed out, "since every irreducible variety in K^n is uniquely determined by one of its generic points, we represent varieties by representing the generic points of their irreducible components. These generic points are given by certain subsets of $K[x_1, \ldots, x_n]$, so-called regular chains in $K[x_1, \ldots, x_n]$". The common roots of any set of multivariate polynomials F (with coefficients in a field) can be decomposed into a finite union of regular chains. Because of the triangular shape of a regular chain, such decomposition is called a *triangular decomposition*.

In computing triangular decompositions, the removal of redundant components is a central issue. In 1987, Wen Tsun Wu [16] introduced the first method for solving systems of algebraic equations by means of so-called "characteristic sets". In his method, many redundant components are produced without any function to remove them. Kalkbrener [10] provided an algorithm where he considered particular characteristic sets, namely regular chains. Although less frequent, the redundant components produced by his algorithm are extremely expensive to remove. At the same time Lazard [11] introduced a practical way to check if a component is contained in another, but these checks may happen quite late in the solving process, so that many intermediate computations can be useless.

The `RegularChains` library employs the algorithm introduced by Moreno Maza [12]. In this algorithm, the computations are managed using a notion of a tree of *tasks*, where redundant branches are easy to cut and can be cut at an early stage. Other than solving systems of algebraic equations symbolically, the `RegularChains` library also provides functionalities for computing modulo regular chains, for instance, polynomial gcds and matrix inverses.

In the following sections we review the mathematical notions of *regular chain* and *triangular decomposition*. Then, we present the `RegularChains` library and illustrate some of its main functionalities: (1) solving polynomial systems symbolically; (2) solving polynomial systems with parameters; (3) computation over non-integral domain; (4) automatic case distinction with recombination; and (5) controlling the properties and the size of the output. We also report on the work-in-progress with the `RegularChains` library, especially regarding the introduction of modular methods for computing triangular decomposition.

2 Regular chains

Let $\mathbb{K}$ be a field and let X be a finite set of variables. Typically, the field $\mathbb{K}$ is the set $\mathbb{Q}$ of the rational numbers or a finite field. Let us call $\mathbb{R}$ the set of the polynomials with coefficients in $\mathbb{K}$ and variables in X. The set X is assumed to be totally ordered. Hence, when looking at a non-constant polynomial p of $\mathbb{R}$, one can talk about its main (or greatest) variable, say v, and the leading coefficient of p w.r.t. v, called the initial of p. We can now describe the shape of a regular chain by defining a more general concept sometimes called *ascending chain* (or *triangular set* in [1]).

A finite set T of non-constant polynomials of $\mathbb{R}$ is an *ascending chain* if two different polynomials of T have different main variables. For instance, if X consists of the variables x and y such that $x > y$ holds, then $[x - y + 1, y^2 + 1]$ is an ascending chain, whereas $[x - y + 1, y^2 - x]$ is not. In broad words, an *ascending chain* is a system of algebraic equations which is ready to solve by evaluating the unknowns one after the other, just like a triangular linear system. However, there is a difference with the linear case: the back solving process may lead to some degenerated situation or even to no solutions. Consider with $x > y$, the ascending chain $[yx - 1, y^2 - y]$. The value $y = 1$ leads to $x = 1$ but the value $y = 0$ does not lead to a value for x. In broad words, regular chains are a particular kind of ascending chains for which the back solving process succeeds in every case.

Regular chains have many interesting computational properties. One of them is that it is very convenient to perform computations modulo a set of relations given by a regular chain. The set of relations which is naturally associated with a regular chain is called its *saturated ideal*. When the regular chain T has as many polynomials as variables, then its saturated ideal is simply the ideal generated by T.

We give now a precise definition of a regular chain and its saturated ideal. First, we recall that a non-zero element h of a ring $\mathbb{A}$ is called *regular* if h is not a zero-divisor, that is, for every f of $\mathbb{A}$, if the product $f\,h$ is null, then f is null. Now, let T be an ascending chain. We define by induction what it means that T is a regular chain. Also we define the saturated ideal of T. If T is empty, then it is a regular chain and its saturated ideal is the trivial ideal (i.e. the ideal consisting only of zero). Assume now that T is not empty. Let p be the polynomial of T with greatest main variable and let $\mathtt{C}$ be the set of the other polynomials in T. If $\mathtt{C}$ is a regular chain with saturated ideal I and if the initial h of p is regular w.r.t. I, then T is a regular chain. In addition, the saturated ideal of T is the set of the polynomials g such that there exists a power h^e of h such that $h^e\,g$ belongs to the ideal generated by I and p. An important property of a regular chain T is that a polynomial f belongs to the saturated ideal of T if and only if f reduces to zero by pseudo-division w.r.t T.

It follows from the previous definition that a set consisting of a single non-constant univariate polynomial q is a regular chain with the ideal generated by q as saturated ideal. Let T be an ascending chain consisting of two polynomials p and q such that q is univariate in y and p is bivariate in x and y. Then T is a regular chain if the gcd of q and the initial of p is 1. This second example generalizes to regular chains with more than two variables or more than two polynomials. Verifying that an ascending chain is a regular chain can be made by means of gcd computations (in the sense of [13]).

These gcd computations take as input two polynomials p_1 and p_2 with the same main variable v and a regular chain T. Since these gcd computations rely on division (or pseudo-division), the initial of the intermediate remainders (or pseudo-remainders) must be regular modulo T. As a consequence, the input polynomials p_1 and p_2 are required to have regular initials, and the output, if it has main variable v, has also an initial regular w.r.t. T.

The saturated ideal of a regular chain is not necessarily prime and working modulo such ideals may involve zero-divisors. However, computing with these zero-divisors is possible following the celebrated D5 Principle, after J. Della Dora, C. Discrescenzo and D. Duval [8]. The idea is the following. When a zero-divisor is encountered during a computation modulo a regular chain, one can split the computations into several cases (or branches) such that in each case this zero-divisor becomes either zero or a regular element (i.e. an element which is not a zero-divisor). Therefore, this zero-divisor is not an obstacle for the computations in any of these branches.

Finally, we would like to mention that several properties relate regular chains to lexicographical Gröbner bases [1, 3]. In particular, strongly normalized regular chains are lexicographical Gröbner bases which provide them with a notion of normal form. Please, refer to [1, 3] for detail on the theoretical aspects of regular chains.

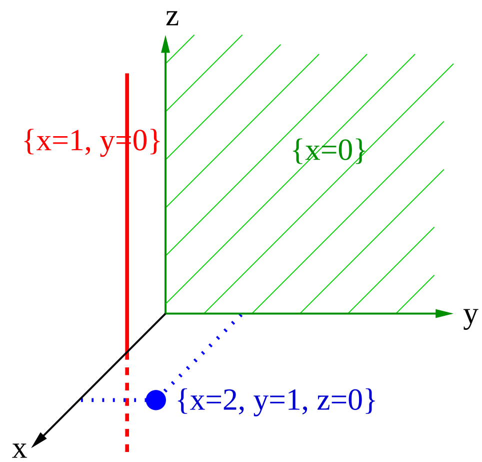

Figure 1: A geometric view of the triangular decomposition of F_1

3 Triangular decompositions

Triangular decompositions are one of the major approaches to solve systems of equations symbolically [16, 10, 11, 15, 12]. For an input system of polynomials F, with rational coefficients, both a Gröbner basis and a triangular decomposition of F give the full set of the complex solutions of F. Consider the polynomial system F_1 with the variables $x > y > z$:

$$\left\{ \begin{array}{lcl} x^3 - 3x^2 + 2x & = & 0 \\ 2yx^2 - x^2 - 3yx + x & = & 0 \\ zx^2 - zx & = & 0 \end{array} \right.$$

It has lexicographical Gröbner basis:

$$\left\{ \begin{array}{lcl} x^2 - xy - x & = & 0 \\ -xy + xy^2 & = & 0 \\ zxy & = & 0 \end{array} \right.$$

and triangular decomposition:

$$\left\{ \; x = 0 \right. \quad \text{or} \quad \left\{ \begin{array}{l} x = 1 \\ y = 0 \end{array} \right. \quad \text{or} \quad \left\{ \begin{array}{l} x = 2 \\ y = 1 \\ z = 0 \end{array} \right.$$

The geometric view of the triangular decomposition of the above polynomial system is illustrated in Figure 1. It is clearly shown that it consists of one point $(x = 2, y = 1, z = 0)$, one line $(x = 1, y = 0)$, and one plane $(x = 0)$;

The triangular decomposition algorithm [12], called `Triade`, relies on a notion of a *task*. A task consists of a set of "algebraic constraints" (namely, a regular chain) and a system of equations to solve under these constraints. A task is called *solved* when its system part is empty; the output of the algorithm is a list of solved tasks. The solving process relies on a procedure called `transform` which takes a non-solved task as input and returns a list of tasks (some of them may be solved, but not necessarily). Therefore, the whole process can be depicted by a tree where every node is a task and where the leaves form the set of solutions of the input problem, i.e. a list of regular chains $C_1, \ldots, C_s$.

The `Triade` top-level procedure looks like:

$toDo := [initialTask]$
$results := [\,]$
while $toDo \neq [\,]$ **repeat**
  $aTask :=$ choose $(toDo)$
  $toDo := toDo \setminus \{aTask\}$
  $newTasksAndResults :=$ transform$(aTask)$
  $(toDo, results) :=$ update$(toDo, results, newTasksAndResults)$
return $results$

One of the most important features of the `Triade` algorithm is that regular chains (i.e. solved tasks) are produced by decreasing order of dimension. This ensures that the redundant components can be removed at an early stage of the solving process. Indeed, the dimension of a redundant component is less than or equal to that of the components in which it is contained.

There are two possible relations between the common roots of F and the regular chains $C_1, \ldots, C_s$, leading to two notions of a triangular decomposition. We say that $C_1, \ldots, C_s$ is a triangular decomposition of F in the sense of *Kalkbrener* if the following holds: a point is a root of F if and only if it is a root of one of the saturated ideals of the $C_1, \ldots, C_s$.

To introduce the other notion of a triangular decomposition we need a definition. We say that a point P is a root of a regular chain T if P cancels every polynomial of T but does not cancel any of the initials of the polynomials of T. We say that $C_1, \ldots, C_s$ is a triangular decomposition of F in the sense of *Lazard* if the following holds: a point is a root of F if and only if it is a root of one of the regular chains $C_1, \ldots, C_s$. In particular, a triangular decomposition in the sense of Lazard is a triangular decomposition in the sense of Kalkbrener. The `RegularChains` library provides both kinds of triangular decompositions. Please, refer to [12] for detail on the theoretical aspects of triangular decompositions.

4 Overview of the `RegularChains` library

The `RegularChains` library is a collection of commands for solving systems of algebraic equations symbolically and studying their solutions. The field $\mathbb{K}$ of coefficients can be $\mathbb{Q}$, a prime field, or a field of multivariate rational functions over $\mathbb{Q}$ or a prime field. In addition, a remarkable feature of the `RegularChains` library is that it also provides functions for computing modulo regular chains, based on the algorithms of [12, 13].

The most frequently used functions are accessible at the top level module in the library. Two submodules, `ChainTools` and `MatrixTools`, contain additional commands to manipulate regular chains and triangular decompositions.

4.1 The top level module

In the top level module of `RegularChains`, the main function is `Triangularize`. For a set F of polynomials, the command `Triangularize` computes the common roots of F in an algebraic closure $\mathbb{L}$ of $\mathbb{K}$ in a form of a triangular decomposition. (If $\mathbb{K}$ is $\mathbb{Q}$, then $\mathbb{L}$ is the field of the complex numbers.) By default, the sense of Kalkbrener is used. An option for solving in the sense of Lazard is available. The operation `PolynomialRing` allows the user to define the polynomial ring $\mathbb{R}$ in which the computations take place, together with the order of the variables.

One very useful function is `RegularGcd`, which computes gcds of two polynomials p_1 and p_2 with common main variable v modulo a regular chain `rc`. It returns a list of pairs $[g_i, rc_i]$ where g_i is a polynomial and rc_i is a regular chain. For each pair, the polynomial g_i is a `gcd` of p_1 and p_2 modulo the saturated ideal of rc_i. Moreover, the leading coefficient of the polynomial g_i w.r.t. v is regular modulo the saturated ideal of rc_i. Finally, the returned regular chains rc_i form a triangular decomposition of `rc` (in the sense of Kalkbrener). See the example in Section 5. Other useful functions are `NormalForm` (which applies only to strongly normalized regular chains) and `SparsePseudoRemainder` (which can be used with any regular chains) for "reducing" a polynomial modulo a regular chain.

4.2 The ChainTools submodule

The `ChainTools` submodule is a collection of commands to manipulate regular chains. These commands split into different categories. First, the commands `Empty`, `ListConstruct`, `Construct`, `Chain` create regular chains from lists of polynomials and other regular chains. Other commands allow to inspect the properties of a regular chain, such as `IsZeroDimensional` and `IsStronglyNormalized`.

The commands `DahanSchostTransform` and `Lift` perform transformation on a regular chain, whereas the commands `EquiprojectableDecomposition`, `SeparateSolutions`, `Squarefree` perform transformation on a triangular decomposition.

The commands `IsInSaturate` and `IsInRadical` compare one polynomial p and one regular chain T. The first one decides whether p belongs to the saturated ideal I of T. This is done without computing a system of generators of I, just by pseudo-division. In fact, the `RegularChains` module never computes explicitly a system of generators for a saturated ideal. The second command decides whether p belongs to the radical of I; this is achieved simply by gcd computations.

The commands `EqualSaturatedIdeals` and `IsIncluded` compare two regular chains T_1 and T_2. More precisely, the first one can decide whether the saturated ideals of T_1 and T_2 are equal or not. If the second command returns true, then the saturated ideal of T_1 is contained in that of T_2. However, if it returns false, nothing can be said.

4.3 The MatrixTools submodule

The `MatrixTools` submodule is a collection of commands to manipulate matrices of polynomials modulo regular chains, including `IsZeroMatrix`, `JacobianMatrix`, `LowerEchelonForm`, `MatrixInverse`, `MatrixMultiply`, and `MatrixOverChain`.

The main purpose of the commands of the MatrixTools submodule is to compute the inverse of the Jacobian matrix of a polynomial system modulo (the saturated ideal of) a regular chain. This question arises for instance in Hensel lifting techniques for triangular sets [14]. The commands of the MatrixTools submodule are quite standard by their goals: multiplication of matrices, computation of inverse or lower echelon of a matrix. However, these commands are considered here in a non-standard context. Indeed, the coefficients of these matrices are polynomials and the computations are performed modulo (the saturated ideal of) a regular chain. In case of a zero-divisor, following the D5 principle [8], the computations split into branches, where in each branch the zero-divisor becomes either zero or a regular element.

5 The RegularChains keynote features

We present here a *tour d'horizon* of the RegularChains library by means of a series of examples. The first ones are for non-experts in symbolic computations whereas the last two require some familiarity with this area.

5.1 Solving polynomial systems symbolically

In this first example, we show how the RegularChains library can solve systems of algebraic equations symbolically. After loading the library in our MAPLE session, we define the ring of the polynomials of the system to be solved. Indeed, most operations of the RegularChains library requires such polynomial ring as argument. This is how one specifies the variable ordering. In our example we choose `x > y > z`. Other arguments passed to the PolynomialRing command could be a set of parameters or the characteristic of the ground field. By default, there are no parameters and the characteristic is zero. Hence, in our example below, the polynomial ring is $\mathbb{Q}[x, y, z]$, that is the ring of polynomials in x, y, z with rational number coefficients.

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

$$R := \mathit{polynomial\_ring}$$

Then we define a set of polynomials of R by

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

$$sys := \{x^2 + y + z - 1, x + y^2 + z - 1, x + y + z^2 - 1\}$$

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

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

$$dec := [\mathit{regular\_chain}, \mathit{regular\_chain}, \mathit{regular\_chain}, \mathit{regular\_chain}]$$

Because regular chains may involve large expressions, one needs ask for viewing them! The command `Equations` displays the list of polynomials of a regular chain. The first three regular chains are very simple: each of them clearly corresponds to a point in the space. Let us have a closer look at the last one. The polynomial in `z` has two solutions. Each of them corresponds to a point in the space.

```
> map(Equations, dec, R);
```

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

Observe that we can also impose inequations. Below we impose the condition $x - z \neq 0$. Then, two points from the original decomposition have been removed.

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

$$decn := [regular\_chain, regular\_chain]$$
$$[[x-1,y,z],[x,y,z-1]]$$

5.2 Solving polynomial systems with parameters

The `RegularChains` library can solve systems of equations with parameters. To illustrate this feature, let us consider a "generic" linear system with 2 unknowns and 2 equations. First we declare `x,y,a,b,c,d,g,h` all as variables.

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

$$sys := \{ax+by-g, cx+dy-h\}$$

In this setting, having more unknowns than equations, our system has an infinite number of solutions. There are two ways of solving such systems. First, by describing its "generic solutions", which is done by computing a triangular decomposition in the sense of Kalkbrener. This is the default behavior of the `Triangularize` command. Observe that the cases where the determinant $-c\,b + a\,d$ vanishes are not explicitly described.

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

$$dec := [regular\_chain]$$
$$[[cx+dy-h,(-cb+ad)\,y+cg-ah]]$$

Now let us compute all the solutions (generic or not) that is a triangular decomposition in the sense of Lazard. To do so, we use the option `output=lazard` of the `Triangularize` command.

```
> dec := Triangularize(sys, R, output=lazard);
```

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

When a regular chain encodes an infinite number of solutions, these solutions are the values canceling any of the polynomials returned by the `Equations` command and none of the polynomials returned by the `Inequations` command. In the command below, for each regular chain of `dec`, we display on the same line its list of equations eq and its list of inequations $ineq$. For instance, the solutions given by the first regular chain in `dec` satisfy simultaneously $cx + dy - h = 0$, $(-cb + ad)y + cg - ah = 0$, $-cb + ad \neq 0$ and $c \neq 0$.

```
> [seq([eq=Equations(dec[i],R), ineq=Inequations(dec[i],R)], i=1..nops(dec))];
```

$$\begin{array}{lllllll}
\mathrm{eq} & = & \{cx + dy - h, (-cb + ad)y + cg - ah\} & ineq & = & \{-cb + ad, c\} \\
\mathrm{eq} & = & \{ax + by - g, dy - h, c\}, & ineq & = & \{a, d\} \\
\mathrm{eq} & = & \{cx + dy - h, -cb + ad, -dg + hb\}, & ineq & = & \{h, c, d\} \\
\mathrm{eq} & = & \{cx - h, -cg + ah, b, d\}, & ineq & = & \{h, c\} \\
\mathrm{eq} & = & \{dy - h, a, -dg + hb, c\}, & ineq & = & \{h, d\} \\
\mathrm{eq} & = & \{cx + dy, -cb + ad, g, h\}, & ineq & = & \{c, d\} \\
\mathrm{eq} & = & \{by - g, a, c, d, h\}, & ineq & = & \{b\} \\
\mathrm{eq} & = & \{x, b, d, g, h\}, & ineq & = & \{\} \\
\mathrm{eq} & = & \{y, a, c, g, h\}, & ineq & = & \{\} \\
\mathrm{eq} & = & \{a, b, c, d, g, h\}, & ineq & = & \{\}
\end{array}$$

Now, we change our polynomial ring in order to specify that `g` and `h` are parameters. This means that we consider now the ring of polynomials in variables `x,y,a,b,c,d` with coefficients in the field of rational functions $\mathbb{Q}(g, h)$. When solving our input system in the sense of Lazard with this new polynomial ring, the last five cases above are discarded since $g = 0$ or $h = 0$ cannot hold anymore.

```
> R2 := PolynomialRing([x,y,a,b,c,d],{g,h}):
> dec := Triangularize(sys, R2, output=lazard):
> [seq([eq=Equations(dec[i],R2), ineq=Inequations(dec[i],R2)], i=1..nops(dec))];
```

$$\begin{array}{lllllll}
\mathrm{eq} & = & \{cx + dy - h, (-cb + ad)y + cg - ah\} & ineq & = & \{-cb + ad, c\} \\
\mathrm{eq} & = & \{ax + by - g, dy - h, c\}, & ineq & = & \{a, d\} \\
\mathrm{eq} & = & \{cx + dy - h, -cb + ad, -dg + hb\}, & ineq & = & \{c, d\} \\
\mathrm{eq} & = & \{cx - h, -cg + ah, b, d\}, & ineq & = & \{c\} \\
\mathrm{eq} & = & \{dy - h, a, -dg + hb, c\}, & ineq & = & \{d\}
\end{array}$$

To summarize:

- one can specify (in advance) a set of variables to be viewed as parameters (this was done with the latter call to `Triangularize` obtaining 5 cases)

- or one can discover the largest set of variables which can be viewed as parameters (this was done with the first call to `Triangularize`, leading to the generic points, in the sense of Kalkbrener)

- or one can view all variables as unknowns (as in the second call to `Triangularize`, returning 10 cases).

5.3 Computation over non-integral domains

The `RegularChains` library provides linear algebra and polynomial computations over towers of simple extensions. These algebraic structures, which appear naturally when solving polynomial systems, may possess zero-divisors. Below, we construct a regular chain `rc` with two simple algebraic extensions. The first one is the extension of the field of rational numbers by $\sqrt{2}$. The second one is not a field extension and introduces *zero-divisors*. Indeed, its defining polynomial $y^2 - y + x - 2$ factorizes as $(y - x)(y + x - 1)$ modulo the defining polynomial $x^2 - 2$ of the first extension.

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

$$R := polynomial\_ring$$

```
> rc := Chain([x^2-2, y^2 -y + x -2], Empty(R), R);
```

$$rc := regular\_chain$$

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

$$[y^2 - y + x - 2, x^2 - 2]$$

Let us compute the gcd of polynomials `p1` and `p2` below w.r.t. `rc`. The example is made such that *splitting* is needed.

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

$$p1 := (y - x)\, z + (y + x - 1)\,(z + 1)$$

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

$$p2 := y - x + (y + x - 1)\,(z + 1)$$

```
> g:= RegularGcd(p1,p2,z,rc,R,normalized=yes);
```

$$g := [[2\,y + zy + zx - z - 1, regular\_chain], [3\,y^2 - 2\,yx - 2\,y - x^2 + 2\,x, regular\_chain]]$$

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

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

```
> rc2 := g[2][2]:  Equations(rc2, R);
```

$$[y + x - 1, x^2 - 2]$$

We obtain two cases. This case discussion comes from the following fact. Modulo the regular chain `rc1`, the `gcd` of `p1` and `p2` has degree 1 w.r.t. `z`, whereas it has degree 0 modulo `rc2`. In general, the output of a gcd computation w.r.t. a regular chain is a list of "cases". Indeed, such gcd is computed by applying the *D5 principle*.

5.4 Automatic case distinction with recombination

Below, we compute the inverse (when it exists) of two matrices `m` modulo a regular chain `rc`. As in the previous example, this regular chain defines a tower of simple extensions with zero-divisors. Each matrix `m` leads to two cases. The first matrix `m` is invertible in one case, but not in the other. The second matrix `m` is invertible in both cases and the corresponding answers are recombined. This recombination feature of the `RegularChains` library is a by-product of the notion of an *equiprojectable decomposition* recently introduced in [5]. In addition, this example illustrates the capabilities of the `RegularChains` library for solving linear systems over non-integral domains.

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

$$R := \mathit{polynomial\_ring}$$

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

$$rc := \mathit{regular\_chain}$$

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

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

```
> m := Matrix([[1, y+z], [0, y-z]]);
```

$$m := \begin{bmatrix} 1 & y+z \\ 0 & y-z \end{bmatrix}$$

```
> MatrixInverse(m,rc,R);
```

$$[[[\begin{bmatrix} 1 & 0 \\ 0 & 1/2\,z^3 \end{bmatrix}, \mathit{regular\_chain}]], [[\text{“}\mathit{no\ Inverse}\text{''}, \begin{bmatrix} 1 & y+z \\ 0 & y-z \end{bmatrix}, \mathit{regular\_chain}]]]$$

```
> m := Matrix([[1, y+z], [2, y-z]]);
```

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

```
> mim := MatrixInverse(m,rc,R);
```

$$\mathit{mim} := [[[\begin{bmatrix} 1 & 0 \\ -z^3 & 1/2\,z^3 \end{bmatrix}, \mathit{regular\_chain}], [\begin{bmatrix} 0 & 1/2 \\ -1/2\,z^3 & 1/4\,z^3 \end{bmatrix}, \mathit{regular\_chain}]], []]$$

```
> m1 := mim [1][1][1]:  rc1 := mim [1][1][2]:  Equations(rc1, R);
```

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

```
> m2 := mim [1][2][1]:  rc2 := mim [1][2][2]:  Equations(rc2, R);
```

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

```
> mc := MatrixCombine([rc1, rc2], R, [m1, m2]);
```

$$mc := [[\begin{bmatrix} 1/2\,z^3y + 1/2 & -1/4\,z^3y + 1/4 \\ -3/4\,z^3 + 1/4\,z^2y & 3/8\,z^3 - 1/8\,z^2y \end{bmatrix}, \mathit{regular\_chain}]]$$

```
> Equations(mc[1][2], R);
```

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

5.5 Controlling the properties and the size of the output

Solving systems of equations by means of regular chains can help reducing the size of the coefficients in the output. Even when no splitting arises! On the example below, due to Barry Trager, we compare the size of the output of `Triangularize` with the lexicographical Gröbner basis for the same variable ordering. We do not print this Gröbner basis nor the regular chain, and we print only their size (as number of characters in the output).

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

$$R := polynomial\_ring$$

```
> sys := [-x^5 + y^5 -3*y -1, 5*y^4 -3, -20*x + y -z];
```

$$sys := [-x^5 + y^5 - 3\,y - 1, 5\,y^4 - 3, -20\,x + y - z]$$

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

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

```
> length(convert(map(Equations,dec,R),string));
```

$$654$$

```
> gb := Groebner:-gbasis(sys,plex(x,y,z)):
> length(convert(gb,string));
```

$$8672$$

On the contrary of the polynomial set `gb`, the regular chain `dec[1]` is not a reduced Gröbner basis of the input system. However, the set `gb` is a regular chain and can be obtained as such by using the option `normalized=yes` of `Triangularize`. In addition, it is possible to obtain from this normalized regular chain (also called "triangular set" in [4, 5, 11]) a more compact regular chain using the *transformation* of Dahan and Schost [4], as shown below. Again, we only show the sizes.

```
> dec := Triangularize(sys, R, normalized=yes);
```

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

```
> length(convert(map(Equations,dec,R),string));
```

$$8674$$

```
> dec2 := map(DahanSchostTransform, dec, R);
```

$$dec2 := [regular\_chain]$$

```
> length(map(Equations,dec2,R),string);
```

$$1692$$

6 Work in progress

The research undertaken in [5, 6, 7] aims at developing fast algorithms and modular method (such as Hensel lifting) to solve systems of polynomials by way of triangular decomposition.

Among all possible triangular decompositions, a canonical one, introduced in [5] and called the *equiprojectable decomposition*, is well suited for modular computations. The function `EquiprojectableDecomposition` in the `RegularChains` library is implemented based on the *split-and-merge* algorithm reported in [6].

The work in [7] shows the high potential to obtain a quasi-linear time (w.r.t. the degree of the algebraic variety defined by the input system F) *split-and-merge* algorithm. A modular algorithm using `Hensel lifting` for triangular decompositions of zero dimensional varieties has been developed in [6]. A preliminary implementation of it shows significant improvements in running time and allows us to solve more difficult problems. It will be available in the next release of the `RegularChains` library.

The next step is to extend these techniques to specialize variables as well during the triangularize modular phase, following the approach initiated in [9] for primitive element representations.

Acknowledgment

The authors would like to thank W. Wu (Univ. of Western Ontario, Canada), É. Schost, X. Dahan (École Polytechnique, France) for their significant contributions to the `RegularChains` library. They are also grateful to H. Ding, X. Li, X, Jin, S. Liang, R. Scott, A. Shakoori, W. Zhou, J. Zhao who have been using the `RegularChains` library for course projects at the University of Western Ontario.

7 Conclusions

The `RegularChains` library provides routines for computing (polynomial GCDs, inverses, ...) modulo regular chains. In particular, this includes computing over towers of field extensions (algebraic or transcendental). In general, this allows computing modulo any radical polynomial ideal, since the operation `Triangularize` can decompose any such ideal into regular chains. The `RegularChains` library is distributed in MAPLE 10. New developments will be included in the next release.

References

[1] P. Aubry and D. Lazard and M. Moreno Maza. On the Theories of Triangular Sets. *J. Symb. Comp.*, 28:105–124, 1999.

[2] Thomas Becker and Volker Weispfenning. *Gröbner Bases: a computational approach to commutative algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer Verlag, 1991.

[3] F. Boulier and F. Lemaire. Computing canonical representatives of regular differential ideals. In proc. *ISSAC 2000*, St Andrews, ACM Press, 2000.

[4] X. Dahan and É. Schost. Sharp Estimates for Triangular Sets. In proc. *ISSAC 04*, Santander, ACM Press, 2004.

[5] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Equiprojectable decomposition of zero-dimensional varieties. In proc. *ICPSS*, Paris, 2004.

[6] X. Dahan, M. Moreno Maza, É. Schost, W. Wu and Y. Xie. Lifting techniques for triangular decompositions. In proc. *ISSAC'05*, 2005.

[7] X. Dahan, M. Moreno Maza, É. Schost, W. Wu and Y. Xie. On the complexity of the D5 principle. submitted poster, *ISSAC'05*, 2005.

[8] J. Della Dora, C. Discrescenzo and D. Duval. About a new Method for Computing in Algebraic Number Fields. In Proc. *EUROCAL 85* Vol. 2, Springer-Verlag, 1985.

[9] M. Giusti, J. Heintz, J. E. Morais, and L. M. Pardo. When polynomial equation systems can be solved fast? In *AAECC-11*, pages 205–231. Springer, 1995.

[10] M. Kalkbrener. A generalized Euclidean algorithm for computing triangular representations of algebraic varieties. *J. Symb. Comp.*, 15:143–167, 1993.

[11] D. Lazard. Solving zero-dimensional algebraic systems. *J. Symb. Comp.*, 13:117–133, 1992.

[12] M. Moreno Maza. On triangular decompositions of algebraic varieties. MEGA-2000 Conference, Bath, 2000.

[13] M. Moreno Maza and R. Rioboo. Polynomial Gcd Computations over Towers of Algebraic Extensions. Proc. AAECC-11, Springer, 1995.

[14] É. Schost. Degree Bounds and Lifting Techniques for Triangular Sets. Proc. ISSAC 2002, 238–245, Teo Mora, jul, ACM Press, 2002.

[15] D. M. Wang. An elimination method for polynomial systems. *J. Symb. Comp.*, 16: 83–114, 1993.

[16] W. T. Wu. A zero structure theorem for polynomial equations solving. *MM Research Preprints*, 1:2–12, 1987.