

Overview of the RegularChains Package

Calling Sequence

RegularChains[**command**](**arguments**)
command(**arguments**)

Description

The **RegularChains** package is a collection of commands for solving systems of algebraic equations symbolically and studying their solutions.

The main function is [Triangularize](#). This function computes from a system of polynomials S a list of simpler systems S_1, \dots, S_n such that a point is a solution of S if and only if it is a solution of one of the systems S_1, \dots, S_n . Each of these simpler systems S_1, \dots, S_n is called a *regular chain* since it has a special shape and remarkable properties that we shall explain now.

To understand what a regular chain is, you first need to define the input system S . It is assumed to be a list (or a set) of polynomials with coefficients in a field K and with variables from a set X . Typically, the field K is the set of the rational numbers. Call R the set of the polynomials with coefficients in K and variables in X . The set X is assumed to be totally ordered. Hence, when looking at a non-constant polynomial p of R , one can talk about its main (or greatest) variable, say v , and the leading coefficient of p with respect to v , called the *initial* of p .

Now we can describe the shape of a regular chain by defining a more general concept, sometimes called an ascending chain or triangular set. A finite set T of non-constant polynomials of R is a triangular set if two different polynomials of T have different main variables. For example, if X consists of the two variables x and y such that $y > x$ holds, then $[x^2 - 1, y^2 - 1]$ is a triangular set, whereas $[x^2 - 1, y^2 - x]$ is not. In broad words, a triangular set is a system of algebraic equations that is ready to be solved 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 for example, for $y > x$, the triangular set $\{y^2 - 1, y^2 - x\}$. The value $y = 1$ leads to $x = 1$, but the value $y = 0$ does not lead to a value of x . In broad words, regular chains are a particular kind of triangular sets for which the back solving process succeeds in every case. A precise definition of a regular chain is given below, just before the examples.

Regular chains have many interesting computational properties. One property is that it is very convenient to perform computations modulo a set of relations given by a regular chain. The set of relations that is naturally associated with a regular chain is called its saturated ideal. This notion is defined precisely below, just before the examples. When the regular chain T has as many polynomials as variables in X , then its saturated ideal is simply the ideal generated by T . The operations [NormalForm](#) and [SparsePseudoRemainder](#) are used intensively for computing modulo regular chains.

The [MatrixTools](#) subpackage provides commands for solving linear systems of equations modulo the saturated ideal of a regular chain. Among other operations are computations of matrix inverses and lower echelon forms. 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. Since this latter is not required to be a prime ideal, the commands of this subpackage

allows you to do linear algebra computations over non–integral domains.

The [ConstructibleSetTools](#) subpackage provides a large set of commands for manipulating constructible sets. Constructible sets are the fundamental objects of Algebraic Geometry, and they play there the role that ideals play in Polynomial Algebra. In broad terms, a constructible set is the solution set of a system of polynomial equations and inequations. Constructible sets appear naturally in many questions, from high–school problems to advanced research topics.

The [SemiAlgebraicSetTools](#) subpackage contains a collection of commands for isolating and counting real roots of zerodimensional regular chains (that is regular chains with a finite number of complex solutions). It also offers a function performing partial cylindrical algebraic decomposition sampling. Several inspection functions on semi–algebraic sets are also provided. They are intended to support the command [RealRootClassification](#).

The [ParametricSystemTools](#) subpackage provides commands for solving systems of equations that depend on parameters. Given a parametric polynomial system F , this subpackage can be used to answer questions such as: for which values of the parameters does F have solutions? finitely many solutions? N real solutions, for a given N ?

The [ChainTools](#) subpackage provides advanced operations on regular chains. Most of these commands allow you to inspect, construct and transform regular chains, or to check the properties of a polynomial with respect to a regular chain. Some commands operate transformations on a set of regular chains; they can be used to analyze the results computed by the command [Triangularize](#).

The [FastArithmeticTools](#) subpackage contains a collection of commands for computing with regular chains in prime characteristic using asymptotically fast algorithms. Most of the underlying polynomial arithmetic is performed at C level and relies on (multi–dimensional) Fast Fourier Transform (FFT). This imposes some constraints on the characteristic. One of the main purposes of this subpackage is to offer efficient basic routines in order to support the implementation of modular algorithms for computing with regular chains and algebraic numbers.

In addition to its main function [Triangularize](#) and its subpackages [ChainTools](#), [MatrixTools](#), [ConstructibleSetTools](#), [ParametricSystemTools](#), [SemiAlgebraicSetTools](#) and [FastArithmeticTools](#) the **RegularChains** package provides basic commands for computing with polynomials and regular chains. The commands [PolynomialRing](#), [DisplayPolynomialRing](#), [MainVariable](#), [Initial](#), [MainDegree](#), [Rank](#), [Tail](#), and [Separant](#) allow the user to manipulate polynomials in the context of regular chains. The commands [Equations](#) and [Inequations](#) allow the user to inspect a regular chain. The commands [RegularGcd](#), [ExtendedRegularGcd](#), [Inverse](#), [IsRegular](#), [RegularizeInitial](#), [NormalForm](#), [SparsePseudoRemainder](#), and [MatrixCombine](#) provide computations modulo regular chains.

List of the RegularChains Package Commands

The following is a list of available top–level commands. See the appropriate subpackage page for lists of commands in the [ChainTools](#), [ConstructibleSetTools](#), [FastArithmeticTools](#), [MatrixTools](#), [ParametricSystemTools](#) and [SemiAlgebraicSetTools](#) subpackages.

ChainTools	ConstructibleSetTools	DisplayPolynomialRing	Equations
ExtendedRegularGcd	FastArithmeticTools	Inequations	Info

Initial	Inverse	IsRegular	MainDegree
MainVariable	MatrixCombine	MatrixTools	NormalForm
ParametricSystemTools	PolynomialRing	Rank	RegularGcd
RegularizeInitial	SemiAlgebraicSetTools	Separant	SparsePseudoRemainder
Tail	Triangularize		

To display the help page for a particular **RegularChains** command, see [Getting Help with a Command in a Package](#).

Mathematical notions

Here is a precise definition of a regular chain and its saturated ideal.

First, recall that a non-zero element \mathbf{h} of a ring \mathbf{R} is called *regular* if \mathbf{h} is not a zero-divisor; that is, for every \mathbf{f} of \mathbf{R} , if the product $\mathbf{f}\mathbf{h}$ is null, then \mathbf{f} is null.

Now, let \mathbf{T} be a triangular set. We define by induction what it means for \mathbf{T} to be a *regular chain*. Also, we define the *saturated ideal* of \mathbf{T} . If \mathbf{T} is empty, then it is a regular chain and its saturated ideal is the trivial ideal (the ideal consisting only of zero). Assume now that \mathbf{T} is not empty. Let \mathbf{p} be the polynomial of \mathbf{T} with greatest main variable and let \mathbf{C} be the set of the other polynomials in \mathbf{T} . If \mathbf{C} is a regular chain with saturated ideal \mathbf{I} , and if the initial \mathbf{h} of \mathbf{p} is regular with respect to \mathbf{I} , then \mathbf{T} is a regular chain. In addition, the saturated ideal of \mathbf{T} is the set of the polynomials \mathbf{g} such that there exists a power \mathbf{h}^e of \mathbf{h} such that $\mathbf{h}^e * \mathbf{g}$ belongs to the ideal generated by \mathbf{I} and \mathbf{p} . An important property of a regular chain \mathbf{T} is that a polynomial \mathbf{f} belongs to the saturated ideal of \mathbf{T} if and only if \mathbf{f} reduces to zero by pseudo-division with respect to \mathbf{T} . The pseudo-division of a polynomial with respect to a regular chain is implemented by the command [SparsePseudoRemainder](#).

It follows from the previous definition that a set consisting of a single polynomial \mathbf{p} is a regular chain whose saturated ideal is the ideal generated by the primitive part of \mathbf{p} regarded as a univariate polynomial in its main variable.

Let \mathbf{T} be a triangular set consisting of two polynomials \mathbf{p} and \mathbf{q} such that \mathbf{q} is univariate in \mathbf{y} and \mathbf{p} is bivariate in \mathbf{x} and \mathbf{y} . Let \mathbf{h} be the initial of \mathbf{p} . Then \mathbf{T} is a regular chain if the GCD of \mathbf{q} and \mathbf{h} is $\mathbf{1}$. This second example generalizes to regular chains with more than two variables or more than two polynomials. Verifying that a triangular set is a regular chain can be made by means of GCD computations.

These GCD computations take as input two polynomials $\mathbf{p1}$ and $\mathbf{p2}$ with the same main variable \mathbf{v} and a regular chain \mathbf{T} . Since these GCD computations rely on division (or pseudo-division), the initial of the intermediate remainders (or pseudo-remainders) must be regular modulo the saturated ideal of \mathbf{T} . As a consequence, the input polynomials $\mathbf{p1}$ and $\mathbf{p2}$ are required to have regular initials, and the output polynomial, if it has main variable \mathbf{v} , also has an initial regular with respect to \mathbf{T} . This explains the name of the command [RegularGcd](#). You can check that the input polynomials are valid input by calling [IsRegular](#) on their initials. If one of the initials of the input polynomials $\mathbf{p1}$ and $\mathbf{p2}$ is not regular with respect to (the saturated ideal of) \mathbf{T} , then you can split \mathbf{T} into several regular chains T_1, \dots, T_s such that [RegularGcd](#) can be called on $\mathbf{p1}$ and $\mathbf{p2}$ for each of $\mathbf{T1}, \dots, \mathbf{T_s}$. This splitting is obtained by using the [RegularizeInitial](#) command on $\mathbf{p1}$ and $\mathbf{p2}$.

Let \mathbf{K} be a field and let \mathbf{R} a polynomial ring over \mathbf{K} obtained by the command [PolynomialRing](#). When \mathbf{R} has no parameters, the field \mathbf{K} can be \mathbf{Q} , the field of rational numbers, or a prime field. When \mathbf{R} has parameters, the field \mathbf{K} is a field of rational functions. For a set (or a list) \mathbf{F} of polynomials of \mathbf{R} , the command [Triangularize](#) computes the common roots of \mathbf{F} in an algebraically closed field \mathbf{L} containing \mathbf{K} . If \mathbf{K} is \mathbf{Q} , then one can think of \mathbf{L} as the field of the complex numbers. The command [Triangularize](#) returns a list of regular chains C_1, \dots, C_s , which is called a *triangular decomposition* of the common roots of \mathbf{F} .

There are two possible relations between the common roots of \mathbf{F} and the regular chains $\mathbf{C1}, \dots, \mathbf{Cs}$, leading to two notions of a triangular decomposition.

We say that $\mathbf{C1}, \dots, \mathbf{Cs}$ is a triangular decomposition of \mathbf{F} *in the sense of Kalkbrener* if the following holds: a point is a root of \mathbf{F} if and only if it is a root of one of the saturated ideals of $\mathbf{C1}, \dots, \mathbf{Cs}$.

To introduce the other notion of a triangular decomposition, we need a definition. A point \mathbf{P} is a root of a regular chain \mathbf{T} if \mathbf{P} cancels every polynomial of \mathbf{T} but does not cancel any of the initials of the polynomials of \mathbf{T} . The commands [Equations](#) and [Inequations](#) applied to \mathbf{T} return the list of its polynomials and the list of their initials, respectively.

We say that $\mathbf{C1}, \dots, \mathbf{Cs}$ is a triangular decomposition of \mathbf{F} *in the sense of Lazard* if the following holds: a point is a root of \mathbf{F} if and only if it is a root of one of the regular chains $\mathbf{C1}, \dots, \mathbf{Cs}$. A triangular decomposition in the sense of Lazard is in particular a triangular decomposition in the sense of Kalkbrener. But the converse is false.

The command [Triangularize](#) is capable of computing both kinds of triangular decompositions. This is achieved by means of options. By default, the sense of Kalkbrener is used. The command [Triangularize](#) admits other options that allow the user to control the properties of the computed regular chains. One important property is that of being *strongly normalized*; see [ChainTools](#) for the definition of this notion. Indeed, if \mathbf{T} is a strongly normalized regular chain, then you can compute the [NormalForm](#) of a polynomial with respect to \mathbf{T} .

An irreducible univariate polynomial over \mathbf{K} defines both a field extension of \mathbf{K} and a regular chain. More generally, let $\mathbf{L1}$ be a direct product of fields, and let \mathbf{p} be a univariate polynomial over $\mathbf{L1}$ that generates a radical ideal; then \mathbf{p} defines an extension $\mathbf{L2}$ of $\mathbf{L1}$ which is another direct product of fields. It turns out that regular chains are a way to encode extensions of fields or extensions of direct products of fields. This idea is central to the algorithms that the commands [IsRegular](#), [Inverse](#), [RegularizeInitial](#), [RegularGcd](#), and [ExtendedRegularGcd](#) implement. The fact that direct products of fields admit zero-divisors is handled by the celebrated D5 principle, which allows us to extend algorithms working fields to direct products of fields.

The theory of regular chains is based on a recursive and univariate vision of polynomials which reduces computations with multivariate polynomials to series of computations with univariate polynomials. The commands [MainVariable](#), [Initial](#), [MainDegree](#), [Rank](#), [Tail](#), and [Separant](#) are the basic operations in this recursive and univariate vision of multivariate polynomials.

Examples

Presented here is an overview of the **RegularChains** library by means of a series of examples. The first ones are for non-experts in symbolic computations, whereas the last ones require some familiarity with this area.

Solving polynomial systems by means of regular chains

The first example shows how the **RegularChains** library can solve systems of algebraic equations symbolically. Start by loading the library.

```
with(RegularChains) : with(ChainTools) : with(MatrixTools) :
```

First, define the ring of the polynomials of the system to be solved. Indeed, most operations of the **RegularChains** library require such a polynomial ring as a parameter. This is how to specify the variable ordering. See [PolynomialRing](#) for more details.

```
R := PolynomialRing([x, y, z])
R := polynomial_ring (5.1.1)
```

Define a set of polynomials of \mathbf{R} .

```
sys := {x^2 - y - z + 1, x - y^2 - z + 1, x - y - z^2 + 1}
sys := {x - y - z^2 + 1, x - y^2 - z + 1, x^2 - y - z + 1} (5.1.2)
```

Ideally, we would like to decompose the solutions of this system into a list of points. In broad terms, this is what the command [Triangularize](#) does. However, some of these points are grouped because they share some properties. These groups are described by regular chains.

```
dec := Triangularize(sys, R)
dec := [regular_chain, regular_chain, regular_chain, regular_chain] (5.1.3)
```

Because these points may involve large expressions, you need to ask to see them! The command [Equations](#) displays the list of polynomials of a regular chain.

```
map(Equations, dec, R)
[[x - 1, y, z], [x, y - 1, z], [x, y, z - 1], [x - z, y - z, z^2 - 2z - 1]] (5.1.4)
```

The first three regular chains are very simple: each of them clearly corresponds to a point in the space. Have a closer look at the last one. The polynomial in z has two solutions. To each of them corresponds a point in the space. You can retrieve these five points by using the solve command.

```
solve(sys)
{x = 1, y = 0, z = 0}, {x = RootOf(_Z^2 - 2_Z - 1), y = RootOf(_Z^2 - 2_Z - 1), z = RootOf(_Z^2 - 2_Z - 1)}, {x = 0, y = 1, z = 0}, {x = 0, y = 0, z = 1} (5.1.5)
```

Consider again the regular chain above that corresponds to two points. Since these two points are grouped together, you can check whether each of them is a solution of the input system. This can be achieved by means of the command [IsInRadical](#) from the [ChainTools](#) subpackage by using the following fact: a regular chain \mathbf{T} encodes a subset of the solution set of the input system \mathbf{S} if and only if every polynomial of \mathbf{S} belongs to the radical of the saturated ideal of \mathbf{T} .

```
seq(IsInSaturate(sys_i, dec_i, R), i = 1 .. nops(sys))
```

$$\text{true, true, true} \quad (5.1.6)$$

Note that `Triangularize` can also take inequations among its input. Below, impose the condition that x must be different from z .

$$\begin{aligned} \text{decn} &:= \text{Triangularize}(\text{sys}, [x \neq z], R); \text{map}(\text{Equations}, \text{decn}, R) \\ \text{decn} &:= [\text{regular_chain}, \text{regular_chain}] \\ &[[x \neq 1, y, z], [x, y, z \neq 1]] \end{aligned} \quad (5.1.7)$$

Observe that two points from the original decomposition have been removed.

If you are solving with the `lazard` option, then the output is a constructible set, rather than a list of regular chains. Such a distinction matters, if the input system is in positive dimension.

$$\begin{aligned} \text{cs} &:= \text{Triangularize}([\text{op}(1..2, \text{sys})], [x \neq z], R, \text{output} = \text{lazard}); \text{Info}(\text{cs}, R) \\ &+ + \quad \text{cs} := \text{constructible_set} \quad + + \\ > \quad [[x \neq y \neq z^2 \neq 1, y^2 \neq z \neq y \neq z^2], [y \neq 1 \neq z^2 \neq z]] \end{aligned} \quad (5.1.8)$$

$$\begin{aligned} \text{decn} &:= \text{Triangularize}([\text{op}(1..2, \text{sys})], [x \neq z], R); \text{map}(\text{Equations}, \text{decn}, R) \\ &+ \text{decn} := [\text{regular_chain}^+] \\ &[[x \neq y \neq z^2 \neq 1, y^2 \neq z \neq y \neq z^2]] \end{aligned} \quad (5.1.9)$$

By default, `Triangularize` gives generic solutions. With the `lazard` option, it outputs all solutions, which generally form a constructible set.

Computing inverses modulo a regular chain

The second example illustrates an important feature of the `RegularChains` library: computations modulo a regular chain. To do so, consider a second system.

$$\begin{aligned} \text{sys} &:= \{x^8 - yx^2 - x - z, x^2y - 3x + 2, x^2 - yx + z\} \\ > \quad \text{sys} := \{x^2 - xy - z, x^2y - 3x - 2, x^8 - x^2y - x - z\} \end{aligned} \quad (5.2.1)$$

$$\begin{aligned} \text{dec} &:= \text{Triangularize}(\text{sys}, R) \\ \text{dec} &:= [\text{regular_chain}] \end{aligned} \quad (5.2.2)$$

The solution computed by the `Triangularize` command consists of a single regular chain. In this polynomial ring, the variable ordering makes $x > y > z$. This means that the x -coordinate of each point must be expressed in terms of y and z , and the y -coordinate as a function of z .

$$\begin{aligned} \text{rc} &:= \text{dec}_1 \\ > \quad \text{rc} := \text{regular_chain} \end{aligned} \quad (5.2.3)$$

$$\text{pz} := \text{Polynomial}(z, \text{rc}, R) \quad (5.2.4)$$

$$pz := z^9 \quad 22 z^8 \quad 208 z^7 \quad 1126 z^6 \quad 3834 z^5 \quad 8136 z^4 \quad 9053 z^3 \quad 224 z^2 \quad 14055 z \quad 13302 \quad (5.2.4)$$

$$py := \text{Polynomial}(y, rc, R) + \dots + \dots + \dots + \dots$$

$$py := 5832 \quad 10056 y \quad 21096 z \quad 21282 z^2 \quad 4056 y z^2 \quad 3177 z^4 \quad 11093 z^3 \quad (5.2.5)$$

$$px := \text{Polynomial}(x, rc, R) + \dots + \dots$$

$$px := 2 \quad 3x \quad x y^2 \quad y z \quad (5.2.6)$$

The polynomial **py** gives **y** as a rational function in **z**. You may ask if you could express **y** as a polynomial function in **z**. In other words, can we replace **py** by a polynomial with **1** as its Initial? Indeed, the polynomial in **z** defines a field extension **K** of the field **Q** of the rational numbers. In the field **K**, you can compute the Inverse of the initial of **py**.

$$\text{newrc} := \text{Under}(y, rc, R); \text{Equations}(\text{newrc}, R)$$

$$+ \dots + \dots + \dots \quad \text{newrc} := \text{regular_chain}$$

$$[z^9 \quad 22 z^8 \quad 208 z^7 \quad 1126 z^6 \quad 3834 z^5 \quad 8136 z^4 \quad 9053 z^3 \quad 224 z^2 \quad 14055 z \quad 13302] \quad (5.2.7)$$

$$lcy := \text{Initial}(py, R); \text{ilcy} := \text{Inverse}(lcy, \text{newrc}, R) + \dots + \dots$$

$$lcy := 10056 \quad 4056 z^2 \quad 9416 z \quad 42 z^4 \quad 16 z^5 \quad 742 z^3 \quad 4 z^6$$

$$\text{ilcy} := [[[143594593667712002746022313 + 157661907907876621441394914 z \quad (5.2.8)$$

$$58395516864169985401373466 z^4 + 31177496499998617409832000 z^3$$

$$+ 118245600582993390444999143 z^2 + 6829798380079214157184280 z^6$$

$$+ 28121353959653668128495638 z^5 \quad 43056952878224602831649 z^8$$

$$851556715329089933233928 z^7, 100270485333918213150443394312,$$

$$> \text{regular_chain}]], []]$$

$$\text{nilcy} := \text{ilcy}_{1_1}; \text{dilcy} := \text{ilcy}_{1_2}$$

$$\text{nilcy} := 143594593667712002746022313 \quad 157661907907876621441394914 z$$

$$58395516864169985401373466 z^4 + 31177496499998617409832000 z^3$$

$$+ 118245600582993390444999143 z^2 + 6829798380079214157184280 z^6$$

$$+ 28121353959653668128495638 z^5 \quad 43056952878224602831649 z^8$$

$$851556715329089933233928 z^7$$

$$dilcy := 100270485333918213150443394312 \quad (5.2.9)$$

The help page for [Inverse](#) explains how to read the output of this command. In this example, this output means that the inverse of **lcy** is a fraction with numerator **nilcy** and denominator **dilcy**. To check that **ilcy** is the inverse of **lcy** modulo **newrc**, use the command [NormalForm](#) to simplify the product **nilcy*****lcy**. Regular chains have a notion of normal form attached to them, just like Groebner bases.

$$\begin{aligned} & \text{NormalForm}(nilcy \text{ lcy}, newrc, R) \\ & 100270485333918213150443394312 \end{aligned} \quad (5.2.10)$$

You obtain **dilcy** as expected. Now you can make the polynomial **py** look better by multiplying it by **ilcy** and removing its content.

$$\begin{aligned} & newpy := \text{NormalForm}(nilcy \text{ py}, newrc, R) : cnewpy := \text{content}(newpy) \\ & cnewpy := 8391661701681509748 \end{aligned} \quad (5.2.11)$$

$$\begin{aligned} newpy & := \frac{newpy}{cnewpy} \\ newpy^+ & := 11948823594 y^+ 78024336215 z^+ 54918900470 z^2 + 54553636695 z^4 \\ & + 88747638462 z^3 19396993429 z^5 559158565 z^7 4309096681 z^6 \\ & 31071832 z^8 156442784340 \end{aligned} \quad (5.2.12)$$

The same treatment can be applied to the polynomial **px**.

$$\begin{aligned} & newrc := \text{Chain}([newpy], newrc, R); \text{Equations}(newrc, R) \\ & newrc^+ := regular_chain^+ \\ & [11948823594 y 78024336215 z 54918900470 z^2 54553636695 z^4 \end{aligned} \quad (5.2.13)$$

$$\begin{aligned} & + 88747638462 z^3 19396993429 z^5 559158565 z^7 4309096681 z^6 \\ & + 31071832 z^8 156442784340, z^9 22 z^8 208 z^7 1126 z^6 3834 z^5 \\ & > 8136 z^4 9053 z^3 224 z^2 14055 z 13302] \end{aligned}$$

$$\begin{aligned} & lcx := \text{Initial}(px, R); ilcx := \text{Inverse}(lcx, newrc, R) \\ & ilcx := 3 y^2 \end{aligned} \quad (5.2.14)$$

$$\begin{aligned} ilcx & := [[[1492395674188006257 1137261498645304126 z \\ & 280077377390474863 z^2 914048962864580664 z^3 \\ & 628167256520883558 z^4 233680431492813710 z^5 \\ & 52074812760151232 z^6 6472881536029244 z^7 336484613006303 z^8, \\ & 822063553694174988, regular_chain], []] \end{aligned}$$

$$nilcx := ilcx_{1_1}; dilcx := ilcx_{1_2} +$$

$$\begin{aligned}
 nilcx := & 1492395674188006257 \quad 1137261498645304126 \quad z \\
 & 280077377390474863 \quad z^2 \quad 914048962864580664 \quad z^3 \\
 & 628167256520883558 \quad z^4 \quad 233680431492813710 \quad z^5 \\
 & 52074812760151232 \quad z^6 \quad 6472881536029244 \quad z^7 \quad 336484613006303 \quad z^8 \\
 > & dilcx := 822063553694174988 \qquad \qquad \qquad (5.2.15)
 \end{aligned}$$

$$newpx := NormalForm(nilcx \text{ p}, newrc, R); cnewpx := content(newpx)$$

$$\begin{aligned}
 newpx := & 822063553694174988 \quad x \quad 4280831462806439198 \quad z \\
 & 867887371605727550 \quad z^2 \quad 2350486446974102010 \quad z^4 \\
 & 3377744504245671336 \quad z^3 \quad 872326306307566684 \quad z^5 \\
 & 23382929946795160 \quad z^7 \quad 192607789095942472 \quad z^6 \\
 & 1173512875760890 \quad z^8 \quad 5386815096525691500 \\
 > & cnewpx := 68798702 \qquad \qquad \qquad (5.2.16)
 \end{aligned}$$

$$newpx := \frac{newpx}{cnewpx} +$$

$$\begin{aligned}
 newpx := & 11948823594 \quad x \quad 62222561449 \quad z \quad 12614880025 \quad z^2 \quad 34164691755 \quad z^4 \quad (5.2.17) \\
 & 49096049868 \quad z^3 \quad 12679400642 \quad z^5 \quad 339874580 \quad z^7 \quad 2799584636 \quad z^6 \\
 & 17057195 \quad z^8 \quad 78298208250
 \end{aligned}$$

Then you obtain a new regular **newrc** chain which encodes the same solution set as **rc**.

$$newrc := Chain([newpx], newrc, R)$$

$$> newrc := regular_chain \qquad \qquad \qquad (5.2.18)$$

$$polys := Equations(rc, R) \quad + \quad + \quad + \quad +$$

$$\begin{aligned}
 polys := & [(3 \quad y^2) \quad x \quad 2 \quad y \quad z, (10056 \quad 4056 \quad z^2 \quad 9416 \quad z \quad 42 \quad z^4 \quad 16 \quad z^5 \\
 & + 742 \quad z^3 \quad 4 \quad z^6) \quad y \quad 5832 \quad + \quad 21096 \quad z \quad 21282 \quad z^2 \quad 3177 \quad z^4 \quad 11093 \quad z^3 \\
 & + 426 \quad z^5 \quad 3 \quad z^7 \quad 6 \quad z^6, \quad z^9 \quad 22 \quad z^8 \quad 208 \quad z^7 \quad 1126 \quad z^6 \quad 3834 \quad z^5 \quad 8136 \quad z^4 \\
 > & 9053 \quad z^3 \quad 224 \quad z^2 \quad 14055 \quad z \quad 13302] \quad (5.2.19)
 \end{aligned}$$

$$newpolys := Equations(newrc, R)$$

$$newpolys := [11948823594 \quad x \quad 62222561449 \quad z \quad 12614880025 \quad z^2 \quad (5.2.20)$$

```

34164691755 z^4 49096049868 z^3 12679400642 z^5 339874580 z^7
2799584636 z^6 17057195 z^8 78298208250, 11948823594 y
+ 78024336215 z 54918900470 z^2 + 54553636695 z^4 88747638462 z^3
19396993429 z^5 + 559158565 z^7 + 4309096681 z^6 31071832 z^8
+ 156442784340, z^9 22 z^8 208 z^7 1126 z^6 3834 z^5 8136 z^4
> 9053 z^3 224 z^2 14055 z 13302 ]
seq(IsInSaturate(polys_i, newrc, R), i = 1 .. nops(polys))
> true, true, true (5.2.21)
seq(IsInSaturate(newpolys_i, rc, R), i = 1 .. nops(newpolys))
true, true, true (5.2.22)

```

This new regular **newrc** has an additional property with respect to **rc**: it is strongly normalized. See [IsStronglyNormalized](#) for the definition of strongly normalized. Being strongly normalized is a requirement in order to use the command [NormalForm](#).

You could have obtained this regular chain from the input system by using an option of [Triangularize](#).

```

decn := Triangularize(sys, R, normalized = yes)
> decn := [regular_chain] (5.2.23)
map(Equations, decn, R)
[[ [11948823594 x 62222561449 z 12614880025 z^2 34164691755 z^4 (5.2.24)
49096049868 z^3 12679400642 z^5 339874580 z^7 2799584636 z^6
+ 17057195 z^8 78298208250, 11948823594 y 78024336215 z
+ 54918900470 z^2 + 54553636695 z^4 88747638462 z^3 19396993429 z^5
+ 559158565 z^7 + 4309096681 z^6 31071832 z^8 156442784340, z^9
22 z^8 208 z^7 1126 z^6 3834 z^5 8136 z^4 9053 z^3 224 z^2
14055 z 13302 ]]]

```

The **Triangularize** command does not always return the normalized decomposition so that it can handle the most general cases, including very large examples. Indeed, observe that the first regular chain **rc** has smaller coefficients than the strongly normalized regular chain **newrc**.

Automatic case discussion

The next example shows that **RegularChains** can handle automatic case discussion. Start by trying

to answer the following question. Why does the above output of [Inverse](#) look complicated? Because **RegularChains** can handle automatic case discussion! To illustrate this, consider two variables y and z ; assume that they are solutions of the regular chain below.

$$\begin{array}{l} R := \text{PolynomialRing}([y, z]) \\ \text{>} \qquad \qquad \qquad R := \text{polynomial_ring} \end{array} \quad (5.3.1)$$

$$\begin{array}{l} rc := \text{Empty}(R) \\ \text{>} \qquad \qquad \qquad + \qquad \qquad \qquad rc := \text{regular_chain} \end{array} \quad (5.3.2)$$

$$\begin{array}{l} \text{>} \quad rc := \text{Chain}([z^4 - 1, y^2 - z^2], rc, R) : \\ \text{Equations}(rc, R) \qquad \qquad \qquad + \\ \qquad \qquad \qquad \qquad \qquad \qquad [y^2 - z^2, z^4 - 1] \end{array} \quad (5.3.3)$$

Compute the inverse of the following matrix modulo the relations of the regular chain rc .

$$\begin{array}{l} m := \text{Matrix}([[1, y - z], [0, y - z]]) \\ \qquad \qquad \qquad + \\ \qquad \qquad \qquad m := \begin{bmatrix} 1 & y & z \\ 0 & y & z \end{bmatrix} \end{array} \quad (5.3.4)$$

Clearly, the result depends on whether y and z are equal or not.

$$\begin{array}{l} mim := \text{MatrixInverse}(m, rc, R) \\ \qquad \qquad \qquad + \\ mim := \left[\left[\left[\left[\begin{array}{cc} 1 & 0 \\ 0 & \frac{1}{2} z^3 \end{array} \right], \text{regular_chain} \right] \right], \left[\left[\text{"noInv"}, \left[\begin{array}{cc} 1 & y & z \\ 0 & y & z \end{array} \right], \text{regular_chain} \right] \right] \end{array} \quad (5.3.5)$$

Check the first result.

$$\begin{array}{l} m1 := mim_{11} \\ \qquad \qquad \qquad + \\ \qquad \qquad \qquad m1 := \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} z^3 \end{bmatrix} \end{array} \quad (5.3.6)$$

$$\begin{array}{l} \text{>} \quad rc1 := mim_{12} \\ \text{>} \qquad \qquad \qquad rc1 := \text{regular_chain} \end{array} \quad (5.3.7)$$

$$\begin{array}{l} \text{Equations}(rc1, R) \qquad \qquad \qquad + \qquad \qquad \qquad + \\ \qquad \qquad \qquad \qquad \qquad \qquad [y - z, z^4 - 1] \end{array} \quad (5.3.8)$$

MatrixMultiply(m1, m, rc1, R)

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

(5.3.9)

> Consider now the other matrix.

m := Matrix([[1, y z], [2, y z]]) +

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

(5.3.10)

mim := MatrixInverse(m, rc, R)

$$mim := \left[\left[\left[\begin{bmatrix} 1 & 0 \\ z^3 & \frac{1}{2} z^3 \end{bmatrix}, \text{regular_chain} \right], \left[\begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} z^3 & \frac{1}{4} z^3 \end{bmatrix}, \text{regular_chain} \right], \right. \right. \quad (5.3.11)$$

[]

> Double check.

m1 := mim₁₁

$$m1 := \begin{bmatrix} 1 & 0 \\ z^3 & \frac{1}{2} z^3 \end{bmatrix}$$

(5.3.12)

rc1 := mim₁₂

rc1 := regular_chain

(5.3.13)

m2 := mim₂₁

$$m2 := \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} z^3 & \frac{1}{4} z^3 \end{bmatrix}$$

(5.3.14)

$rc2 := mim_{1_2 2}$

> $rc2 := regular_chain$ (5.3.15)

$MatrixMultiply(m2, m, rc2, R)$

> $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (5.3.16)

$MatrixMultiply(m2, m, rc2, R)$

> $+ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (5.3.17)

Can you get a "generic" answer that would hold both cases? Yes, you can.

$clr := MatrixCombine([rc1, rc2], R, [m1, m2])$

$clr := \left[\begin{array}{cccc} \frac{1}{2} y z^3 & \frac{1}{2} & \frac{1}{4} y z^3 & \frac{1}{4} \\ \frac{3}{4} z^3 & \frac{1}{4} y z^2 & \frac{3}{8} z^3 & \frac{1}{8} y z^2 \end{array} \right], regular_chain$ (5.3.18)

Check.

$MatrixMultiply(clr_{1_1}, m, clr_{1_2}, R)$

> $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (5.3.19)

Recombining the results from a case discussion

The overview of the **RegularChains** library continues with more advanced examples, extending on the topic of automatic case discussion.

Can you have several cases in the output of [MatrixCombine](#)? Yes, this can happen. Reuse the first polynomial system above.

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

> $+ + + R := polynomial_ring$ (5.4.1)

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

> $sys := \{x y z^2 + 1, x y^2 z + 1, x^2 y z + 1\}$ (5.4.2)

$lrc := Triangularize(sys, R, normalized = yes); map(Equations, lrc, R)$

$$lrc := [regular_chain, regular_chain, regular_chain, regular_chain]$$

$$[[x^2 - 1, y, z], [x, y^2 - 1, z], [x, y, z^2 - 1], [x^2 - z, y^2 - z, z^2 - 2z - 1]] \quad (5.4.3)$$

Generate four random matrices.

> *randomize*(4869257127) :

$$lm := [seq(Matrix([seq(seq(randpoly([x, y, z], degree = 1), j = 1..2)], i = 1..2)], k = 1..4)]$$

$$lm := \begin{bmatrix} 73x & 52y & 18z & 93 & 32x & 96y & 4z & 90 \\ 78x & 63y & 12z & 29 & 11x & 41y & 41z & 94 \end{bmatrix}, \quad (5.4.4)$$

$$\begin{bmatrix} 60x + 12y & 75z + 11 & 18x & 66y & 74z + 22 \\ 27x & 90y & 49z & 16 & 98x + 20y + 76z + 93 \end{bmatrix},$$

$$\begin{bmatrix} 17x & 78z & 70 & 48x & 48y & 43z & 68 \\ 67x + 94y + 90z & 86 & 28x & 67y & 68z & 52 \end{bmatrix},$$

$$\begin{bmatrix} 64x & 15y & 84z & 71 & 65x & 29y & 29z & 65 \\ 46x & 34y & 61z & 82 & 20x & 33y & 6z & 57 \end{bmatrix}$$

Now ask for the re-combination of the four cases.

$$clr := MatrixCombine(lrc, R, lm)$$

$$clr := \left[\begin{bmatrix} 166 & 189y & 122 & 166y \\ 107 & 33y & 83 & 10y \end{bmatrix}, regular_chain \right], \quad (5.4.5)$$

$$\left[\begin{bmatrix} 29 & 42z^2 & 79z & \frac{77}{2} & \frac{53}{2}z^2 & 46z \\ \frac{315}{2} & \frac{151}{2}z^2 & 78z & 68 & 11z^2 & 41z \end{bmatrix}, regular_chain \right]$$

It turns out that you cannot obtain a unique case. This is surprising, since the saturated ideals of the four regular chains are pairwise relatively prime.

Check why the re-combination into a single regular chain is not possible.

$$rcl := clr_{12}$$

$$rcl := regular_chain \quad (5.4.6)$$

$$\begin{aligned} & \text{Equations}(rc1, R) + \\ > \quad [x \ y \ 1, y^2 \ y, z] \end{aligned} \quad (5.4.7)$$

$$rc2 := clr_{2_2}$$

$$> \quad rc2 := regular_chain \quad (5.4.8)$$

$$\begin{aligned} & \text{Equations}(rc2, R) + \quad + \quad + \quad + \\ & [2x \ z^2 \ 1, z^2 \ 2y \ 1, z^3 \ z^2 \ 3z \ 1] \end{aligned} \quad (5.4.9)$$

The two ideals generated by **rc1** and **rc2** are obviously relatively prime (no common roots in **z**). But if you try to recombine them, you create a polynomial **qy** in **y** with a zero-divisor as initial; this is forbidden by the properties of a regular chain. Construct the polynomial **qy** and check if its initial is a zero-divisor.

$$\begin{aligned} & Rz := PolynomialRing([z]) \\ > \quad Rz := polynomial_ring \end{aligned} \quad (5.4.10)$$

$$\begin{aligned} & rc := Empty(Rz) \\ > \quad rc := regular_chain \end{aligned} \quad (5.4.11)$$

$$\begin{aligned} & rc1 := Chain([z], rc, Rz) \\ > \quad + \quad rc1 := regular_chain \end{aligned} \quad (5.4.12)$$

$$\begin{aligned} & rc2 := Chain([z^3 \ z^2 \ 3z \ 1], rc, Rz) \\ > \quad rc2 := regular_chain \end{aligned} \quad (5.4.13)$$

$$\begin{aligned} & m1 := Matrix([[y^2 \ y]]) \\ > \quad + \quad m1 := \begin{bmatrix} y^2 & y \end{bmatrix} \end{aligned} \quad (5.4.14)$$

$$\begin{aligned} & m2 := Matrix([[2y \ z^2 \ 1]]) \\ > \quad m2 := \begin{bmatrix} z^2 & 2y & 1 \end{bmatrix} \end{aligned} \quad (5.4.15)$$

$$\begin{aligned} & clr := MatrixCombine([rc1, rc2], Rz, [m1, m2]); m := clr_{1_1} \\ & clr := \quad + \quad + \quad + \quad + \\ & \quad \left(\begin{bmatrix} z^3 y^2 & 3y z^3 & z^2 y^2 & 3y z^2 & 3z y^2 & 9y z & y^2 & y & z^3 & 3z \\ 2z^2 \end{bmatrix}, regular_chain \right) \\ & m := \begin{bmatrix} z^3 y^2 & 3y z^3 & z^2 y^2 & 3y z^2 & 3z y^2 & 9y z & y^2 & y & z^3 & 3z & 2z^2 \end{bmatrix} \end{aligned} \quad (5.4.16)$$

```
rc := clr12; qy := m1,1
```

$$\begin{aligned}
 & + \quad rc := \text{regular_chain} \quad + \quad + \quad + \\
 > qy := z^3 y^2 \quad 3y z^3 \quad z^2 y^2 \quad 3y z^2 \quad 3z y^2 \quad 9yz \quad y^2 \quad y \quad z^3 \quad 3z \quad 2z^2 \quad (5.4.17)
 \end{aligned}$$

```
Inverse(Initial(qy, R), rc, Rz)
```

$$\begin{aligned}
 & [[[1, 1, \text{regular_chain}], [\text{regular_chain}, \text{regular_chain}]]] \quad (5.4.18)
 \end{aligned}$$

Solving systems with an infinite number of solutions

The next example in the overview of **RegularChains** is a second round for experts. All previous automatic case discussions involve discussions with algebraic numbers only. Can the **RegularChains** library handle automatic case discussion with parameters? Yes, this is possible.

Consider the following system.

```
R := PolynomialRing([x, y, a, b, c, d, g, h])
```

$$\begin{aligned}
 > \quad + \quad +R := \text{polynomial_ring} \quad (5.5.1)
 \end{aligned}$$

```
sys := {a x - b y - g, c x - d y + h}
```

$$\begin{aligned}
 & sys := \{a x - b y - g, c x - d y - h\} \quad (5.5.2)
 \end{aligned}$$

This new system has a property that the previous examples do not have. Clearly, this new system has an infinite number of solutions, if we view its 8 variables as unknowns. 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.

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

```
+ dec := [regular_chain]
```

$$\begin{aligned}
 & [[c x - d y - h, (c b - a d) y - c g - a h]] \quad (5.5.3)
 \end{aligned}$$

Computing triangular decompositions in the sense of Kalkbrener is the default mode of [Triangularize](#).

Observe that the output does not provide explicitly the solutions of the system that cancel the determinant $a*d-b*c$. Now compute all the solutions (generic or not); that is, a triangular

decomposition in the sense of Lazard.

```
dec := Triangularize(sys, R, output = lazar)
```

$$\begin{aligned}
 dec := [\text{regular_chain}, \text{regular_chain}, \text{regular_chain}, \text{regular_chain}, \quad (5.5.4) \\
 \text{regular_chain}, \text{regular_chain}, \text{regular_chain}, \text{regular_chain}, \text{regular_chain}, \\
 > \text{regular_chain}, \text{regular_chain}]
 \end{aligned}$$

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

$$\begin{aligned}
 & [[c x - d y - h, (c b - a d) y - c g - a h], [a x - b y - g, d y - h, c], [c x \\
 & d y - h, c b - a d, d g - h b], [a x - b y - g, c, d, h], [c x - h, c g
 \end{aligned}$$

```

a h, b, d], [d y h, a, d g h b, c], [c x d y, c b a d, g, h], [b y
g, a, c, d, h], [x, b, d, g, h], [y, a, c, g, h], [a, b, c, d, g, h]]
[ {c, c b a d}, {a, d}, {c, d, h}, {a}, {c, h}, {d, h}, {c, d}, {b}, {}, {}, (5.5.5)
> {} ]

```

```

[ seq( [ eq = Equations( dec, R), ineq = Inequations( dec, R) ], i = 1 .. nops( dec) ) ]
[ [ eq = [c x + d y h, ( c b a d) y c g a h], ineq = {c, c b a d}], [ eq (5.5.6)
± [a x b y + g, d y h, c], ineq = {a, d}], [ eq ± [c x d y h, c b
a d, d g h b], ineq = {c, d, h}], [ eq = [a x b y g, c, d, h], ineq
= {a}], [ eq = [c x h, c g a h, b, d], ineq = {c, h}], [ eq = [d y h, a,
d g h b, c], ineq = {d, h}], [ eq = [c x d y, c b a d, g, h], ineq = {c,
d}], [ eq = [b y g, a, c, d, h], ineq = {b}], [ eq = [x, b, d, g, h], ineq
= {} ], [ eq = [y, a, c, g, h], ineq = {} ], [ eq = [a, b, c, d, g, h], ineq = {} ] ]

```

By defining the [PolynomialRing](#) correctly, you find that [Triangularize](#) can solve polynomial systems with parameters.

```

R2 := PolynomialRing( [x, y, a, b, c, d], {g, h} )
> R2 := polynomial_ring (5.5.7)

```

```

dec := Triangularize( sys, R2, output = lazar )
dec := [regular_chain, regular_chain, regular_chain, regular_chain, regular_chain] (5.5.8)

```

```

[ seq( [ eq = Equations( dec, R2), ineq = Inequations( dec, R2) ], i = 1 .. nops( dec) ) ]
[ [ eq = [c x + d y h, ( c b a d) y c g a h], ineq = {c, c b a d}], [ eq (5.5.9)
± [a x b y + g, d y h, c], ineq = {a, d}], [ eq = [c x + d y h, c b
a d, d g h b], ineq = {c, d}], [ eq = [c x h, c g a h, b, d], ineq
= {c}], [ eq = [d y h, a, d g h b, c], ineq = {d} ] ]

```

Similarly, [Triangularize](#) can solve polynomial systems in prime characteristic.

```

R2 := PolynomialRing( [x, y, a, b, c, d], {g, h}, 3 )
> R2 := polynomial_ring (5.5.10)

```

```

dec := Triangularize( sys, R2, output = lazar )
dec := [regular_chain, regular_chain, regular_chain, regular_chain, (5.5.11)
regular_chain ]

```

```
[seq([eq = Equations(dec_i, R2), ineq = Inequations(dec_i, R2)], i = 1..nops(dec))]
```

```
[ [eq = [c x d y 2 h, (2 c b + a d) y c g 2 a h], ineq = {c, 2 c b + a d}], (5.5.12)
```

```
[eq = [a x b y + 2 g, d y 2 h, c], ineq = {a, d}], [eq = [c x + d y 2 h,
```

```
2 c b a d, 2 d g h b], ineq = {c, d}], [eq = [c x 2 h, 2 c g a h, b,
```

```
d], ineq = {c}], [eq = [d y 2 h, a, 2 d g h b, c], ineq = {d}]]
```

Controlling the size of the coefficients

Solving systems of equations by means of regular chains can help reduce the size of the coefficients, even when no splitting arises.

In the example below, compare the size of the output of [Triangularize](#) with the lexicographical Groebner basis for the same variable ordering. Do not print this Groebner basis since it is quite large; print its size (number of characters) only.

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

```
> R := polynomial_ring R (5.6.1)
```

```
sys := { x^5 y^5 3 y 1, 5 y^4 - 3, 20 x y + z }
```

```
> sys := { 5 y^4 3, 20 x y z, x^5 y^5 3 y 1 } (5.6.2)
```

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

```
+ dec := [regular_chain]
```

```
[ [20 x y z, (4375 z12 52800011625 z8 32000000000 z7
```

```
+ 110591902080002925 z4 61439980800000000 z3 1280000000000000 z2
```

```
5662311727104036800027) y 1875 z13 23592963686400144000000
```

```
+ 9600010125 z9 200000000 z8 7372714752004545 z5
```

```
3072000240000000 z4 1280000000000000 z3 22118403456000135 z,
```

```
3125 z20 9375 z16 + 4000000000 z15 2015999988750 z12
```

```
156000000000 z11 1920000000000000 z10 12165125356800006750 z8
```

```
1474560223200000000 z7 652800000000000000 z6
```

```
4096000000000000000000000000 z5 16986908639233347839997975 z4
```

```
14155767152640302400000000 z3 5898238732800000000000000 z2
```

```
1228800000000000000000000000 z 6195303619231982878732441600243 ]]
```

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

```
> 654 (5.6.4)
```

```
> with( Groebner ) :
```

```
gb := Basis( sys, plex( x, y, z ) ) : length( convert( gb, string ) )
```

```
8672 (5.6.5)
```

The commands below illustrate the fact that the **RegularChains** library provides tools to reduce the size of the coefficients of an output. To see this, use the previous system again, starting from a large output. Indeed, it turns out that for the above system, the lexicographical Groebner basis can be obtained also by using [Triangularize](#) with the option **normalize=yes**. This is because every strongly normalized regular chain **T** is a lexicographical Groebner basis over the field of the rational functions in the variables that are not algebraic in **T**. In this example, each variable [IsAlgebraic](#) in the output regular chain.

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

```
> dec := [ regular_chain ] (5.6.6)
```

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

```
8674 (5.6.7)
```

Then, the command [DahanSchostTransform](#) can be applied to reduce this large regular chain (which is also a lexicographical Groebner basis) into a smaller one.

```
dst := DahanSchostTransform( dec1, R )
```

```
> dst := regular_chain (5.6.8)
```

```
length( convert( Equations( dst, R ), string ) )
```

```
1534 (5.6.9)
```

Check that the two regular chains define the same (saturated) ideal by means of the command [EqualSaturatedIdeals](#).

```
EqualSaturatedIdeals( dec1, dst, R )
```

```
true (5.6.10)
```

Splitting for solving

In this library, almost every operation takes a regular chain as a parameter. A regular chain encodes a tower of simple extensions of the underlying field. This tower is a direct product of fields and may contain zero-divisors, so splitting may be needed.

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

```
R := polynomial_ring (5.7.1)
```

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

```
> rc := regular_chain (5.7.2)
```

```
rc := Chain([z (z - 1)], rc, R)
```

```
> rc := regular_chain (5.7.3)
```

```
p1 := z x (x - 1) (z - 1) x (x - 2) +
```

```
> p1 := z x (x - 1) (z - 1) x (x - 2) (5.7.4)
```

```
p2 := z (x - 1) (x - 1) (z + 1) (x - 3) (x - 2) +
```

```
> p2 := z (x - 1) (x - 1) (z - 1) (x - 3) (x - 2) (5.7.5)
```

```
expand(p1) + +
```

```
> z x x^2 2 x (5.7.6)
```

```
expand(p2) + + +
```

```
> 7 z 5 z x x^2 5 x 6 (5.7.7)
```

```
RegularGcd(p1, p2, x, rc+R) +
```

```
> [[ 4 z x 3 x 7 z 6, regular_chain ]] (5.7.8)
```

As a consequence, every operation (taking a regular chain as a parameter) needs to manage tasks, where a task is [something-to-compute, a-regular-chain].

Manipulating Constructible Sets

This example demonstrates how to manipulate constructible sets, which usually encode the solution set of a polynomial system with both equations and inequations.

```
> with(ConstructibleSetTools) :
```

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

```
R := polynomial_ring (5.8.1)
```

Define a polynomial system with equations **F** and inequations **H**.

```
F := [s (y - 1) x, s (x - 1) y]; H := [s - 1]
```

```
F := [s (y - 1) x, s (x - 1) y]
```

```
H := [s - 1] (5.8.2)
```

Use the **GeneralConstruct** command to create a constructible set **cs** to encode its solutions.

```
cs := GeneralConstruct(F, H, R)
```

```
cs := constructible_set (5.8.3)
```

➤ In the **RegularChains** library, **cs** is represented by a list of regular systems.

$$lrs := \text{RepresentingRegularSystems}(cs, R)$$

$$lrs := [\text{regular_system}, \text{regular_system}] \quad (5.8.4)$$

Each regular system is a pair consisting of a regular chain and an inequation given by one or more polynomials.

$$rs := lrs_1$$

$$rs := \text{regular_system} \quad (5.8.5)$$

$$rc := \text{RepresentingChain}(rs, R)$$

$$rc := \text{regular_chain} \quad (5.8.6)$$

$$h := \text{RepresentingInequations}(rs, R)$$

$$h := [s \neq 1] \quad (5.8.7)$$

➤ The library provides the basic set-theoretic operations on constructible sets, including complementation, union, intersection, difference, inclusion test, and more.

$$F2 := [s^2 - (y^2 - 1)x, s^2 - (x^2 + 1)y]; H := [s + 1]$$

$$F2 := [s^2 - (y^2 - 1)x, s^2 - (x^2 - 1)y]$$

$$H := [s - 1] \quad (5.8.8)$$

$$cs2 := \text{GeneralConstruct}(F2, H, R)$$

$$cs2 := \text{constructible_set} \quad (5.8.9)$$

$$\text{Complement}(cs, R)$$

$$\text{constructible_set} \quad (5.8.10)$$

$$\text{Union}(cs, cs2, R)$$

$$\text{constructible_set} \quad (5.8.11)$$

$$\text{Intersection}(cs, cs2, R)$$

$$\text{constructible_set} \quad (5.8.12)$$

$$\text{Difference}(cs, cs2, R)$$

$$\text{constructible_set} \quad (5.8.13)$$

$$\text{IsContained}(cs, cs2, R)$$

$$\text{false} \quad (5.8.14)$$

Besides these, some advanced operations are also provided. See [ConstructibleSetTools](#) for details.

Solving Parametric Polynomial Systems

This tour d'horizon concludes with an illustration of how to solve parametric polynomial systems with comprehensive triangular decomposition.

Let U be the last \mathbf{d} variables of \mathbf{R} , which are regarded as parameters. The output of `ComprehensiveTriangularize(sys, d, R)` consists of two parts. The first part is a pre-comprehensive triangular decomposition S of \mathbf{sys} with respect to the last \mathbf{d} variables of \mathbf{R} . The second part is a list L of pairs; the first item of each pair is a constructible set and the second item is a list of indices (positive integers) such that the union of these constructible sets forms a partition of the projection of $V(\mathbf{sys})$ onto the parameter space. Moreover, for each part (or cell) C , the list of positive integers associated with it gives the positions of the regular chains in S satisfying the following property: for each parameter value u in C , the associated regular chains specialized at u form a triangular decomposition of the input system \mathbf{sys} specialized at u .

```

R := PolynomialRing([x, y, s])
>
+
R := polynomial_ring
(5.9.1)

F := [s^2 - (y - 1)x, s^2 - (x - 1)y]
+
F := [s^2 - (y - 1)x, s^2 - (x - 1)y]
(5.9.2)

```

A comprehensive triangular decomposition of \mathbf{F} with respect to parameter s is as follows.

```

> with(ParametricSystemTools) :
pctd, cells := ComprehensiveTriangularize(F, 1, R)
pctd, cells := [regular_chain, regular_chain, regular_chain], [[constructible_set,
[3, 2]], [constructible_set, [1]]]
(5.9.3)

```

The first part of the output is the pre-comprehensive triangular decomposition of \mathbf{F} , which consists of three regular chains.

```

map(Info, pctd, R)
+
+
+
[[ (y - 1)x - s, y^2 - s - y ], [x - 1, y - 1, s], [x, y, s]]
(5.9.4)

```

The projection of $V(F)$ onto the parameter space has been partitioned into two disjoint constructible sets.

```

> with(ConstructibleSetTools) :
cs1 := cells_1; Info(cs1, R)
cs1 := constructible_set
>
[[s], [1]]
(5.9.5)
cs2 := cells_2; Info(cs2, R)

```

$cs2 := \text{constructible_set}$

$[[], [s]]$

(5.9.6)

Thus, when the parameter value s is in **cs1**, the last two regular chains (after specialization) form a triangular decomposition of $F(s)$. Otherwise, s is in **cs2**, and the first regular chain forms a triangular decomposition of $F(s)$.

References

The theory of regular chains was introduced independently by M. Kalkbrenner in his PhD Thesis: "Three contributions to elimination theory." and by L. Yang and J. Zhang in the paper "Searching dependency between algebraic equations: an algorithm." Regular chains and their related concepts (triangular sets and saturated ideals) are discussed in the paper "On the theories of triangular sets," co-authored by P. Aubry, D. Lazard, and M. Moreno Maza. The paper "When does T equal Sat(T)?" by F. Lemaire, M. Moreno Maza, W. Pan and Y. Xie, contains a more up-to-date summary of these notions.

The algorithms implemented in the **RegularChains** package are mainly taken from the paper "On Triangular Decompositions of Algebraic Varieties" by M. Moreno Maza, available at the author's web page. The other algorithms are taken from the papers "Lifting techniques for triangular decompositions" (X. Dahan, M. Moreno Maza, E. Schost, W. Wu, and Y. Xie), "Change of ordering for regular chains in positive dimension" (X. Dahan, X. Jin, M. Moreno Maza, and E. Schost, 2007), "Comprehensive Triangular Decomposition" (C. Chen, F. Lemaire, O. Golubitsky, M. Moreno Maza, and W. Pan, 2007), and "Efficient Computations of Irredundant Triangular Decompositions with the RegularChains Library" (C. Chen, F. Lemaire, M. Moreno Maza, W. Pan, and Y. Xie, 2007).

The **RegularChains** library has been originally designed by F. Lemaire (Univ. of Lille, France) and M. Moreno Maza (Univ. of Western Ontario, Canada). Today, the design, implementation and maintenance is mainly the work of F. Lemaire, Y. Xie (MIT, Cambridge, USA), and M. Moreno Maza. Large contributions have been made by C. Chen, X. Jin, L. Li, X. Li, E. Schost, W. Pan, and W. Wu (all of them from the Univ. of Western Ontario, Canada) and by B. Xia and R. Xiao (Peking University, China). X. Dahan (Kyushu University, Japan) and H. Ding, Y. Li, A. Shakoori, W. Zhou, and J. Zhao (Univ. of Western Ontario, Canada) have also participated in the development of the **RegularChains** package.

Aubry, P.; Lazard, D. and Moreno Maza, M. "On the theories of triangular sets." **J. Symb. Comp.**, Vol. **28**. (1999): 105–124.

Boulier, F. and Lemaire, F. "Computing canonical representatives of regular differential ideals." **In Proc. ISSAC 2000**. St Andrews, Scotland, 2000.

Boulier, F.; Lemaire, F. and Moreno Maza, M. "Well known theorems on triangular systems and the D5 Principle." **In Proceedings of Transgressive Computing 2006**. Edited by J.–G. Dumas. Spain: University of Granada, 2006.

Chen, C.; Lemaire, F.; Golubitsky, O.; Moreno Maza, M. and Pan, W. "Comprehensive Triangular Decomposition." **Proceedings of CASC 2007, Lecture Notes in Computer Science**, Vol. **4770**. Springer, 2007.

Chen, C.; Lemaire, F.; Moreno Maza, M.; Pan, W. and Xie, Y. "Efficient Computations of Irredundant Triangular Decompositions with the RegularChains Library." **Proceedings of CASA 2007, Lecture**

Notes in Computer Science, Vol. **4488**. Springer, 2007.

Dahan, X.; Moreno Maza, M.; Schost, E. and Xie, Y. "On the complexity of the D5 Principle." In **Proceedings of Transgressive Computing 2006**, Edited by J.–G. Dumas. Spain: University of Granada, 2006.

Dahan, X.; Jin, X.; Moreno Maza, M. and Schost, E. "Change of ordering for regular chains in positive dimension" **J. of Theoretical Computer Science**, to appear, 2007.

Della Dora, J.; Discrescenzo, C. and Duval, D. "About a new method for computing in algebraic number fields." In **Proc. EUROCAL 85** Vol. **2**. 1985.

Kalkbrener, M. "A generalized Euclidean algorithm for computing triangular representations of algebraic varieties." **J. Symb. Comp.**, Vol. **15**. (1993): 143–167.

Kalkbrener, M. "Three contributions to elimination theory." PhD Thesis, University of Linz, Austria, 1991.

Lazard, D. "Solving zero–dimensional algebraic systems." **J. Symb. Comp.**, Vol. **13**. (1992): 117–133.

Lemaire, F.; Moreno Maza, M. and Xie, Y. "The RegularChains library." In **Proceedings of Maple Conference' 05**, pp. 355–368. Edited by I. Kotsireas. 2005.

Lemaire, F.; Moreno Maza, M.; Pan, W. and Xie, Y. "When does T equal Sat(T)?" In **Proc. ISSAC 2008**. Linz, Austria, 2000.

Moreno Maza, M. "On Triangular Decompositions of Algebraic Varieties." **MEGA–2000 conference**. Bath, UK, England.

Moreno Maza, M. and Rioboo, R. "Polynomial GCD computations over towers of algebraic extensions" In *proc. of AAEECC–11*, **Lecture Notes in Comput. Sci**, vol. 948, Springer, 1995.

Moreno Maza, M. and Xie, Y. "Component–level parallelization of triangular decompositions." In **Proceedings of Parallel Symbolic Computation'07**, pp. 69–77, ACM Press, NY, USA, 2007.

Schost, E. "Complexity results for triangular sets." **J. Symb. Comp.**, Vol. **36** No. **34**. (2003): 555–594.

Wang, D. M. "An elimination method for polynomial systems." **J. Symb. Comp.**, Vol. **16**. (1993): 83–114.

Wang, D. M. **Elimination Methods**. Wein, New York: Springer, 2000.

Wu, W. T. "Basic principles of mechanical theorem proving in elementary." **J. Sys. Sci. and Math. Scis**, Vol. **4**. (1984): 207–235.

Yang, L. and Zhang, J. "Searching dependency between algebraic equations: an algorithm." **Artificial intelligence in mathematics**. Oxford University Press, 1994.