# Computing with Constructible Sets in Maple

Changbo Chen [a] François Lemaire [b] Marc Moreno Maza [a]
Liyun Li [a] Wei Pan [a] Yuzhen Xie [c]

[a] *University of Western Ontario*
*Department of Computer Science*
*London, Ontario, Canada N6A 5B7*

[b] *Université de Lille*

[c] *Massachusetts Institute of Technology*

**Abstract**

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. In this paper, we introduce `ConstructibleSetTools` a new module of the `RegularChains` library in `Maple`. To our knowledge, this is the first 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. Simplifying set-theoretical expressions on constructible sets is at the core of fundamental and challenging operations, like the removal of redundant components when decomposing a polynomial system. We present practically efficient approaches for this purpose together with an application to solver verification.

*Key words:* Constructible set, representation, simplification, triangular decomposition, Maple.

## 1. Introduction

Solving systems of equations, algebraic or differential, is a driving subject for symbolic computation. Many practical applications of polynomial system solving require a description of the real solutions of an input system with finitely many complex solutions. Meanwhile, some other applications of polynomial system solving require more

⋆ This research was partly supported by Maplesoft and MITACS of Canada

*Email addresses:* `changbo.chen@gmail.com` (Changbo Chen), `Francois.Lemaire@lifl.fr` (François Lemaire), `moreno@csd.uwo.ca` (Marc Moreno Maza), `liyun@scl.csd.uwo.ca` (Liyun Li), `wpan9@csd.uwo.ca` (Wei Pan), `yxie@csail.mit.edu` (Yuzhen Xie).

advanced operations on ideals and varieties, such as *decomposition into components* (unmixed, irreducible, ...). Primary decomposition of ideals and triangular decomposition of algebraic varieties are concepts which provide the necessary theoretical framework. Algorithms for primary decomposition involve operations on ideals, such as saturation, intersection and quotient computations; their implementation in the computer algebra systems `AXIOM`, `Singular`, `CoCoA`, `MAGMA` and `Maple` has led to packages for computing with polynomial ideals, based on Gröbner basis techniques.

The development of triangular decomposition started in the late 80's with the work of Wu (1987), more than 20 years after the introduction of Gröbner bases by Buchberger (1965). In the early 90's, the notion of a *regular chain*, introduced independently by Kalkbrener (1993) and Yang and Zhang (1991), led to important algorithmic progress and stimulated implementation activity. To our knowledge, computer algebra systems provide solvers based on triangular decompositions for 12 years only, mainly in `Maple`, but also in `AXIOM`, `Singular` and `MAGMA`. Examples of such solvers are the downloadable packages `Epsilon` by D.M. Wang, `WSolve` by D.K. Wang, `DISCOVERER` by B.C. Xia and the `RegularChains` library (Lemaire et al., 2005) shipped with `Maple` since its release 10. Efficient solvers based on triangular decomposition are work in progress (Li et al., 2008).

This implementation effort is supported by continuous theoretical and algorithmic advances. The notion of *comprehensive triangular decomposition* introduced by Chen et al. (2007a) has brought to light the fact that *constructible sets* play the role for triangular decompositions that polynomial ideals play for Gröbner bases. This fact was underlying since the early work of Wu (1984); it became explicit in (Chen et al., 2007a) where the authors provided procedures for computing the set-theoretical difference and intersection of two constructible sets represented by triangular decompositions. Actually, this work motivated the realization of the software presented in this article.

Comprehensive triangular decomposition (CTD) is one of the tools for parametric polynomial system solving, an area which has an increasing number of applications and which is in demand of efficient algorithms and solvers. Of course, the classical techniques based on Gröbner bases and triangular decompositions can process parametric polynomial systems. However, most practical questions related to these systems require specific theoretical and algorithmic enhancements. To highlight this point, let us consider $\Sigma(U, X)$ a parametric polynomial system, where $U$ stands for a set of parameters and $X$ for a set of unknowns. A typical problem is to determine the values of $U$ for which $\Sigma(U, X)$ possesses solutions. This brings the following difficulty: these values of $U$ may not form an algebraic variety, that is, they may not form the solution set of a system of polynomial equations. For instance, for the system $\Sigma(U, X)$ consisting of the single equation $ux - 1 = 0$, with $U = \{u\}$ and $X = \{x\}$, the solution set of our problem is given by $u \neq 0$, which is a constructible set, but not an algebraic variety. Therefore, in the context of a strongly typed language, say `AXIOM`, the implementation of a package for parametric polynomial systems would naturally imply the implementation of a type *constructible_set*. In the case of `Maple`, the implementation of the CTD (in the module `ParametricSystemTools` of the `RegularChains` library) led us to realize a second module, namely `ConstructibleSetTools`.

Let us illustrate this new module with an example from the theory of algebraic curves. Each of the polynomials below defines an elliptic curve in the complex plane of coordinates $(x, y)$:

$$g_1(x, y) = x^3 + ux - y^2 + 1 \text{ and } g_2(x, y) = x^3 + vx - y^2 + 1.$$

They depend on parameters $u$ and $v$ respectively. In invariant theory, a classical question is whether there exists a linear fractional map from the first curve to the second:

$$f : (x, y) \mapsto \left( \frac{a\,x + b\,y + c}{g\,x + h\,y + k}, \frac{d\,x + e\,y + f}{g\,x + h\,y + k} \right)$$

We assume here that the origin is mapped to the origin which sets $c = f = 0$. Writing that the rational function

$$g_1(x, y) - g_2(f(x, y))$$

is identically zero yields a system $F$ with 24 equations (which contain lots of trivial equations), 7 unknowns $a, b, d, e, g, h, k$ and 2 parameters $u, v$. Moreover we must have $(g, h, k) \neq (0, 0, 0)$. Hence, the problem is turned into computing the projection of the set defined by the difference of two varieties $V(F) \setminus V(H)$, where $H$ is the system $\{g, h, k\}$. The command `Difference` of the module `ConstructibleSetTools` can compute such a difference and the output is a constructible set:

$$\left\{ \begin{array}{rcl} a - k & = & 0 \\ b & = & 0 \\ d & = & 0 \\ -k^2 + e^2 & = & 0 \\ h & = & 0 \\ g & = & 0 \\ u - v & = & 0 \\ v & \neq & 0 \\ k & \neq & 0 \end{array} \right. \cup \left\{ \begin{array}{rcl} -ku + va & = & 0 \\ b & = & 0 \\ d & = & 0 \\ -k^2 + e^2 & = & 0 \\ h & = & 0 \\ g & = & 0 \\ u^2 + uv + v^2 & = & 0 \\ v & \neq & 0 \\ k & \neq & 0 \end{array} \right. \cup \left\{ \begin{array}{rcl} a^3 - k^3 & = & 0 \\ b & = & 0 \\ d & = & 0 \\ -k^2 + e^2 & = & 0 \\ h & = & 0 \\ g & = & 0 \\ u & = & 0 \\ v & = & 0 \\ k & \neq & 0. \end{array} \right.$$

The command `Projection` is then used to compute its projection image onto the parameter space defined by $u$ and $v$, which is again a constructible set:

$$\left\{ \begin{array}{rcl} u - v & = & 0 \\ v & \neq & 0 \end{array} \right. \cup \left\{ \begin{array}{rcl} u^2 + uv + v^2 & = & 0 \\ v & \neq & 0 \end{array} \right. \cup \left\{ \begin{array}{rcl} u & = & 0 \\ v & = & 0. \end{array} \right.$$

Such a constructible set can be simplified as $u^3 - v^3 = 0$; interestingly, these calculations lead to another proof of Theorem 1 in (Kogan and Moreno Maza, 2002).

More generally our software allows the user to perform on constructible sets the usual set theoretical operations: union, intersection, difference, complement, and emptiness-test. A more advanced operation is the computation of image (or pre-image) of a constructible set by a rational map: this provides an algorithmic realization of Chevalley's Theorem for constructible sets. Section 3 provides a tour of the `ConstructibleSetTools` module.

In Section 2 we discuss the implementation of this module. We represent a constructible set $C$ by a list $[[T_1, h_1], \ldots, [T_e, h_e]]$ of so-called regular systems, where each $T_i$ is a regular chain and each $h_i$ is a polynomial regular w.r.t. the saturated ideal of $T_i$. Then the points of $C$ are formed by the points that belong to at least one quasi-component $W(T_i)$ without cancelling the associated polynomial $h_i$. This representation reveals important geometrical information such as the degree and dimension of the Zariski closure of $C$. However, it is not canonical: different

regular system decompositions can encode the same constructible set. In order to compare two such decompositions, we propose efficient "simplification" algorithms together with complexity and experimental results.

In Section 4, we give a tour of the commands of the module `ParametricSetTools` dedicated to solving parametric polynomial systems. Another application supported by `ConstructibleSetTools` appears in Section 5: verification of polynomial system solvers.

Up to our knowledge, `ConstructibleSetTools` is the first distributed package for this purpose. Nevertheless, several existing packages have some functionalities related to ours. We attempt to give a survey of those in Section 6.

## 2. Representation, simplification

In algebraic geometry, a constructible set is any zero set of a system of polynomial equations and inequations. Thus each possible representation of such zero sets provides an encoding for constructible sets. Selecting a representation should be guided by the operations to be performed on it; these could be computing intersection, union and differences of constructible sets, etc.

The problems faced there are representative of the usual dilemma of symbolic computation: choosing between *canonical representation* and *expression-tree representation*. Indeed, using a canonical representation, say by means of irreducible varieties as in (Manubens and Montes, 2006b), makes a union computation $C_1 \cup C_2$ non-trivial whereas one could implement the union by just concatenating the representations of $C_1$ and $C_2$. This latter expression-tree approach or *lazy evaluation approach* should be equipped with one (or more) *simplification operation*, for instance, in order to remove duplicated data.

Before presenting the representation used in our software module, we need to review the underlying algebraic notions: *triangular set*, *regular chain*, *regular system* and *constructible set*. For details on these concepts and their properties, please refer to Aubry et al. (1999) and Chen et al. (2007a).

**Triangular set.** Let $\mathbb{K}[X] := \mathbb{K}[x_1, \ldots, x_n]$ be a polynomial ring over a field $\mathbb{K}$ and with ordered variables $x_1 \prec \cdots \prec x_n$. We denote by $\overline{\mathbb{K}}$ the algebraic closure of $\mathbb{K}$. For a set of polynomials $F \subset \mathbb{K}[X]$, we denote by $V(F)$ the *zero set* (or algebraic variety) of $F$ in $\overline{\mathbb{K}}^n$. For a non-constant polynomial $p \in \mathbb{K}[X]$, we denote by $\mathrm{init}(p)$ the leading coefficient of $p$ regarded as a univariate polynomial in its main (or largest) variable. Let $T \subset \mathbb{K}[X]$ be a *triangular set*, that is, a set of non-constant polynomials with pairwise distinct main variables. The *saturated ideal* $\mathrm{sat}(T)$ of $T$ is defined to be the ideal $\langle T \rangle : {h_T}^\infty$, where $h_T$ is the product of initials of polynomials in $T$. The *quasi-component* $W(T)$ of $T$ is $V(T) \setminus V(h_T)$, that is, the set of the zeros of $T$ which do not cancel any initials of $T$.

**Regular chain.** Let $T \subset \mathbb{K}[X]$ be a triangular set. If $T$ is empty, then it is a regular chain. Otherwise, let $p$ be the polynomial of $T$ with greatest main variable and let $C$ be the set of other polynomials in $T$. We say that $T$ is a *regular chain*, if $C$ is a *regular chain* and the initial of $p$ is regular (that is, neither null nor a zero-divisor) modulo $\mathrm{sat}(C)$.

**Regular system.** A pair $[T, h]$ is a *regular system* if $T \subset \mathbb{K}[X]$ is a regular chain, and $h \in \mathbb{K}[X]$ is regular with respect to $\mathrm{sat}(T)$. The zero set of $[T, h]$, denoted by $Z(T, h)$, is defined as $W(T) \setminus V(h)$. Note that the zero set of any regular system $[T, h]$ is not empty; moreover its Zariski closure is equal to $V(\mathrm{sat}(T))$ which is an unmixed algebraic variety.

**Constructible set.** A *constructible set* of $\overline{\mathbb{K}}^n$ is a finite union $(A_1 \setminus B_1) \cup \cdots \cup (A_e \setminus B_e)$ where $A_1, \ldots, A_e, B_1, \ldots, B_e$ are algebraic varieties in $\overline{\mathbb{K}}^n$. In Chen et al. (2007a) it is shown that every constructible set is equal to a finite union of zero sets of regular systems.

Based on this latter result, we represent a constructible set $C$ of $\overline{\mathbb{K}}^n$ by a list of regular systems $[[T_1, h_1], \ldots, [T_e, h_e]]$, with $T_1, \ldots, T_e \subset \mathbb{K}[X]$ and $h_1, \ldots, h_e \in \mathbb{K}[X]$, such that:

$$C = Z(T_1, h_1) \ \cup \ \cdots \ \cup \ Z(T_e, h_e).$$

**Example 1.** For the ordered variable $x \succ y \succ s$ and over the field $\mathbb{Q}$ of rational numbers, the constructible set $C$ given by the conjunction of the conditions

$$s - (y+1)x = 0, \ \ s - (x+1)y = 0, \quad \text{and} \quad s \neq 1$$

is represented by $[R_1, R_2]$ where $R_1 = [T_1, h_1]$ and $R_2 = [T_2, h_2]$ are regular systems with

$$\begin{cases} T_1 \ = \ [(y+1)x - s, y^2 + y - s] \\ h_1 \ = \ s - 1 \end{cases}, \quad \begin{cases} T_2 \ = \ [x+1, y+1, s] \\ h_2 \ = \ 1 \end{cases}.$$

At first glance, our encoding can be regarded as a compromise between a canonical representation and an expression-tree representation. From a decomposition into regular systems, important geometrical information (dimension, degree) can be read directly. Nevertheless, different regular system decompositions may encode the same constructible set. Comparing (for equality test or inclusion test) two such decompositions, however, can be done efficiently. The sequel of this section aims at supporting this claim.

Exploiting the triangular structure of regular systems leads to natural and efficient operations on those objects, see Section 2.1. In Section 2.2 we specify our simplification operations for constructible sets. Sections 2.3, 2.4 and 2.5 provide complexity results, practical algorithms and experimental results for those operations.

### *2.1.* Comparing two regular systems

Our basic tool for manipulating constructible sets is an algorithm for computing the set theoretical difference of the zero sets $Z(T, h)$ and $Z(T', h')$ of two regular systems of $\mathbb{K}[X]$. We introduced this algorithm in (Chen et al., 2007a) as a building block for simplifying the representations of constructible sets by means of the operations `MPD` and `SMPD`, defined in Section 2.2. The objective of the present section is to illustrate the fact that zero sets of regular systems are easy to compare. We also want to stress the fact that these comparisons reduce to polynomial GCD computations modulo regular chains, for which efficient algorithms and implementation are available (Li et al., 2008).

To keep this review simple, we restrict ourselves to the case where $h = h' = 1$ and the initials of $T$ and $T'$ are all equal to 1, too, that is, $h_T = h_{T'} = 1$. A *sketch* of our algorithm computing $Z(T, h) \setminus Z(T', h')$ is given below and is illustrated by Figure 1. We stress the fact that this is only a sketch; in particular, the algorithm makes recursive calls where the input arguments do not satisfy the assumption $h = h' = 1$. The complete algorithm, with proofs, can be found in (Chen et al., 2007a). It has a similar structure as the sketched algorithm below.

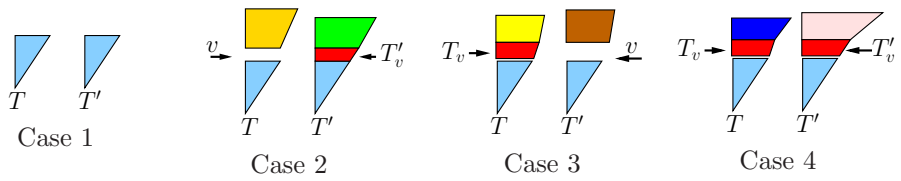


Fig. 1. Computing $Z(T, h) \setminus Z(T', h')$ by exploiting the triangular structure level by level.

*Case 1.* We first check whether the regular chains $T$ and $T'$ generate the same ideal. Under our assumption $h_T = h_{T'} = 1$, this is equivalent to $\mathrm{sat}(T) = \mathrm{sat}(T')$. This latter equality can be tested by pseudo-division, by virtue of Theorem 6.1 in (Aubry et al., 1999). If this equality holds then the difference $Z(T,h) \setminus Z(T',h')$ is empty. (Recall that we assume $h = h' = 1$.) If this equality does not hold, then let $v$ be the largest variable such that $T_{<v}$ and ${T'}_{<v}$ generate the same ideal, where $T_{<v}$ and ${T'}_{<v}$ denote the set of the polynomials with main variable less than $v$ in $T$ and $T'$ respectively. From this point we consider the Cases 2, 3 and 4 below.

*Case 2.* Assume that there is a polynomial $T'_v$ in $T'$ with main variable $v$ and no such a polynomial in $T$. Then the points of $Z(T,h) \setminus Z(T',h') = V(T) \setminus V(T')$ split into two groups:

- those from $V(T)$ that do not cancel $T'_v$, that is, $Z(T, T'_v)$,

- those from $V(T)$ that cancel $T'_v$ but are outside of $V(T')$, that is, $V(T \cup \{T'_v\}) \setminus V(T')$.

*Case 3.* Assume that there is a polynomial $T_v$ in $T$ with main variable $v$ and no such a polynomial in $T'$. Then, it suffices to exclude from $V(T)$ the points of $V(T')$ which cancel $T_v$, that is, $V(T) \setminus V(T' \cup \{T_v\})$.

*Case 4.* Now we assume that both $T_v$ and $T'_v$ are defined. By definition of $v$, they are different modulo the ideal generated by $T_{<v}$. Let $g$ be a GCD of $T_v$ and $T'_v$ modulo $T_{<v}$. (We use here the GCD notion of (Moreno Maza and Rioboo, 1995).) To keep things simple, we assume that the computations do not split and that the initial of $g$ is 1. Three sub-cases arise:

(4.1) If $g$ is a constant then the ideals generated by $T$ and $T'$ are relatively prime, hence $V(T)$ and $V(T')$ are disjoint.

(4.2) If $g$ is non-constant but its main variable is less than $v$, the points of $Z(T,h) \setminus Z(T',h') = V(T) \setminus V(T')$ split into two groups:

- those from $V(T)$ that do not cancel $g$, that is, $Z(T,g)$,
- those from $V(T)$ that cancel $g$ but are still outside of $V(T')$, that is, $V(T \cup \{g\}) \setminus V(T')$.

(4.3) If $g$ has main variable $v$, we just split $T$ following the D5 principle (Della Dora et al., 1985). Hence $T$ is replaced by

$$D_g := T_{<v} \cup \{g\} \cup T_{>v} \quad \text{and} \quad D_q := T_{<v} \cup \{q\} \cup T_{>v}$$

where $q$ is the quotient of $T_v$ by $g$ modulo the ideal generated by $T_{<v}$. Finally, we recursively compute $V(D_g) \setminus V(T')$ and $V(D_q) \setminus V(T')$.

### *2.2.* The `MPD` and `SMPD` operations

When computing with constructible sets, two notions of simplification are equally important to us. The first one is to remove the *redundant components* occurring when concatenating the regular system lists encoding two constructible sets $C_1$ and $C_2$. This happens, obviously, when computing $C_1 \cup C_2$. A formal definition for this simplification follows.

**The `MPD` operation.** Given regular systems $[T_1, h_1], \ldots, [T_e, h_e]$ in $\mathbb{K}[X]$, the function `MakePairwiseDisjoint` (`MPD` for short) returns regular systems $[S_1, g_1], \ldots, [S_f, g_f]$ in $\mathbb{K}[X]$ such that the following conditions hold:

$(i)$ $Z(T_1,h_1) \cup \cdots \cup Z(T_e,h_e) = Z(S_1,g_1) \cup \cdots \cup Z(S_f,g_f)$,
$(ii)$ for all $1 \leq i < j \leq f$ we have $Z(S_i,g_i) \cap Z(S_j,g_j) = \emptyset$.

A second level of redundancy can occur in a family $\mathcal{C}$ of constructible sets $C_1, \ldots, C_m$, where $C_i \cap C_j \neq \emptyset$ might hold for some $1 \leq i < j \leq m$. If each $C_i$ is a meaningful geometrical object (say with some particular properties) replacing $\mathcal{C}$ by another family $\mathcal{D}$ of constructible sets should imply preserving the geometrical information carried by $\mathcal{C}$. This is naturally achieved if $\mathcal{D}$ is an *intersection-free basis* of $\mathcal{C}$, as defined hereafter.

**The `SMPD` operation.** Given a family $\mathcal{C} = \{C_1, \ldots, C_m\}$ of constructible sets of $\overline{\mathbb{K}}^n$, the function `SymmetricallyMakePairwiseDisjoint` (`SMPD` for short) returns a family $\mathcal{D} = \{D_1, \ldots, D_n\}$ of constructible sets of $\overline{\mathbb{K}}^n$ with the following properties:

(*i*) $D_i \cap D_j = \emptyset$ for all $1 \leq i < j \leq n$,
(*ii*) each $C_i$ can be uniquely written as a finite union of some of the $D_j$'s.

We call $\mathcal{D}$ an *intersection-free basis* of $\mathcal{C}$.

The name `SMPD` comes from the idea that the operation `SMPD` must preserve some structure whereas the operation `MPD` is simply meant to remove duplicated data.

### *2.3.* Complexity results

We are interested in analyzing the complexity of algorithms implementing the `MPD` and `SMPD` operations when classical polynomial arithmetic is used for computing GCDs modulo regular chains. Indeed, the current code of our modules `ConstructibleSetTools` and `ParametricSystemTools` does not rely yet on asymptotically fast algorithms, such as FFT-based polynomial arithmetic. In broad terms, our targeted result is the following: if our polynomial arithmetic runs in quadratic time (w.r.t. input sizes) then the operations `MPD` and `SMPD` run "essentially" in quadratic time (w.r.t. input sizes). Theorems 1 and 2 are formal statements of this result. Comments about their proofs appear in Section 2.4 and complete proofs appear in the technical report (Chen et al., 2008).

In our current study, we restrict ourselves to regular systems $[T, h]$ for which the saturated ideal $\mathrm{sat}(T)$ is zero-dimensional. We believe that this special case already allows us to obtain good indication on the performances of our algorithms. Note that for the general case, we have conducted intensive experimental comparisons, see Section 2.5. In future work, we shall relax this zero-dimensional assumption.

Under this hypothesis, the saturated ideal $\mathrm{sat}(T)$ is equal to the ideal generated by $T$; moreover $h$ is invertible modulo $T$ and thus can be assumed to be 1, or equivalently, can be ignored. Finally, we shall assume that the base field $\mathbb{K}$ is perfect and that $\langle T \rangle$ is radical. The latter assumption is easily achieved by squarefree factorization, since $\langle T \rangle$ is zero-dimensional.

**Theorem 1.** Let $T_1, \ldots, T_e$ be zero-dimensional regular chains such that the ideals they generate are radical. Let $d_i$ be the degree of the variety $V(T_i)$, for $1 \leq i \leq e$. Then, there exists a positive constant $K$ (independent of $T_1, \ldots, T_e$) such that, on input $[T_1, 1], \ldots, [T_e, 1]$ the operation `MPD` runs in $O(K^n \sum_{1 \leq i < j \leq e} d_i d_j)$ operations in $\mathbb{K}$.

**Theorem 2.** Let $C_1, \ldots, C_m$ be constructible sets. We assume that each $C_i$ is represented by zero-dimensional regular chains $T_{i,1}, \ldots, T_{i,e_i}$ generating radical ideals; hence we have $C_i = V(T_{i,1}) \cup \cdots \cup V(T_{i,e_i})$. Moreover we assume $V(T_{i,j}) \cap V(T_{i,k}) = \emptyset$ for all $1 \leq j < k \leq e_i$, and we denote by $D_i$ be the number of points in $C_i$, for all $1 \leq i \leq m$. Then, there exists a positive constant $K$ (independent of $C_1, \ldots, C_m$) such that, on input $C_1, \ldots, C_m$, the operation `SMPD` runs in $O(K^n \sum_{1 \leq i < j \leq m} D_i D_j)$ operations in $\mathbb{K}$.

When FFT-based polynomial arithmetic is assumed then the operations `MPD` and `SMPD` run "essentially" in linear time, see (Dahan et al., 2006). However, implementation techniques for those asymptotically fast algorithms remain to be developed.

### 2.4. Algorithms for MPD and SMPD

Our proofs of Theorems 1 and 2 employ the *augment refinement method* by Bach et al. (1990). This has led to algorithms for the operations MPD and SMPD improving those first given in (Chen et al., 2007a). For the sake of conciseness, in the rest of this section, we focus on SMPD. We assume that, given two constructible sets $C_1, C_2$ of $\overline{\mathbb{K}}^n$ we have an operation PairRefine$(C_1, C_2)$ returning a triple $(C_{1\setminus 2}, C_{1\cap 2}, C_{2\setminus 1})$ such that we have

$$C_{1\setminus 2} = C_1 \setminus C_2,\ C_{1\cap 2} = C_1 \cap C_2,\ \text{ and }\ C_{2\setminus 1} = C_2 \setminus C_1.$$

In fact, the "difference" operation sketched in Section 2.1 can be enhanced such that it computes the two differences $Z(T,h) \setminus Z(T',h')$ and $Z(T',h') \setminus Z(T,h)$ together with the intersection $Z(T,h) \cap Z(T',h')$. Algorithms 1 and 2 below both implement the SMPD operation: The first follows the pattern proposed by Bach et al. (1990) for computing GCD-free basis. The second one is a *divide-and-conquer* transformation of Algorithm 1 which leads to interesting experimental results, see Section 2.5.

**Algorithm 1** SMPD

**Input:** a list $L$ of constructible sets

**Output:** an intersection-free basis of $L$

$m \leftarrow |L|$
**if** $m < 2$ **then**
  **output** $L$
**else**
  $I \leftarrow \emptyset;\ D' \leftarrow \emptyset;\ d \leftarrow L[m]$
  $L^* \leftarrow$ SMPD$(L[1, \ldots, m-1])$
  **for** $l \in L^*$ **do**
    $(d, i, d') \leftarrow$ PairRefine$(d, l)$
    $I \leftarrow I \cup i;\ D' \leftarrow D' \cup d'$
  **end for**
  **output** $d \cup I \cup D'$
**end if**

**Algorithm 2** SMPD

**Input:** a list $L$ of constructible sets

**Output:** an intersection-free basis of $L$

$m \leftarrow |L|$
**if** $m < 2$ **then**
  **output** $L$
**else**
  $z \leftarrow \lfloor m/2 \rfloor;\ I \leftarrow \emptyset$
  $L_1 \leftarrow$ SMPD$(L[1, \ldots, z])$
  $L_2 \leftarrow$ SMPD$(L[z+1, \ldots, m])$
  **for** $j$ in $1..|L_1|$ **do**
    **for** $k$ in $1..|L_2|$ **do**
      $(L_1[j], i, L_2[k]) \leftarrow$
        PairRefine$(L_1[j], L_2[k])$
      $I \leftarrow I \cup i$
    **end for**
  **end for**
  **output** $L_1 \cup I \cup L_2$
**end if**

Observe that Algorithms 1 and 2 do not require that constructible sets are represented by means of regular systems. The only requirement is to have at hand a function $(C_1, C_2) \longmapsto$ PairRefine$(C_1, C_2)$ with the above specification. The work in Hong (1992) suggests that the techniques from multiple-valued logic minimization could help with simplifying the computations before calling operations on regular systems (or Gröbner bases).

### 2.5. Experimental results

In this section we provide benchmarks on the implementation of three different algorithms for realizing the SMPD operation in Maple, respectively the original algorithm of Chen et al. (2007a) (we call it Algorithm 0), Algorithm 1 and Algorithm 2. We examine their efficiency by comparing their running times during the computation of a comprehensive triangular decomposition (CTD). Table 1 gives the timing for twenty examples selected from Chen et al. (2007a) (all examples

are in positive dimension). These benchmarks are performed in `Maple` 12 on an Intel Pentium 4 machine (3.20GHz CPU, 2.0GB memory).

| Sys | Algo. 0 | Algo. 1 | Algo. 2 | Sys | Algo. 0 | Algo. 1 | Algo. 2 |
|---|---|---|---|---|---|---|---|
| 9 | 3.817 | 0.818 | 1.112 | 19 | 0.733 | 0.444 | 0.211 |
| 10 | 1.138 | 0.223 | 0.281 | 20 | 0.020 | 0.013 | 0.013 |
| 11 | 12.302 | 3.494 | 0.786 | 21 | 3.430 | 0.584 | 0.633 |
| 12 | 10.114 | 0.383 | 0.318 | 22 | 25.413 | 8.292 | 9.530 |
| 13 | 1.268 | 0.318 | 0.362 | 23 | 1097.291 | 82.468 | 122.575 |
| 14 | 0.303 | 0.103 | 0.062 | 24 | 11.828 | 0.930 | 0.985 |
| 15 | 1.123 | 0.259 | 0.271 | 25 | 54.197 | 1.934 | 1.778 |
| 16 | 2.407 | 1.184 | 0.703 | 26 | 0.530 | 0.047 | 0.064 |
| 17 | 0.574 | 0.091 | 0.159 | 27 | 27.180 | 13.705 | 4.626 |
| 18 | 0.548 | 0.293 | 0.283 | 28 | ¿ 10,000 | 1838.927 | 592.554 |

**Table 1** Timing (s) of `SMPD` algorithms during CTD computation

Given a list $\mathcal{C}$ of constructible sets, Algorithm 0 first collects all their defining regular systems into a list, then computes its intersection-free basis $\mathcal{G}$ which consists of regular systems; finally one can easily deduce from $\mathcal{G}$ an intersection-free basis of $\mathcal{C}$. In this manner the defining regular systems of each constructible set are made (symmetrically) pairwise disjoint, though sometimes this is unnecessary. As reported in Chen et al. (2007a), Algorithm 0 is expensive and sometimes can be a bottleneck.

Our benchmark results suggest that algorithms 1 and 2 are practically more efficient than Algorithm 0. This has led to improve the performances of our CTD code in a significant manner. For instance, it can solve now Sys 28, posted by Lazard in ASCM 2001, which our previous implementation could not process. Besides, Algorithm 2 performs more than 3 times faster than Algorithm 1 for some examples: it probably behaves better w.r.t. cache locality due to its *divide-and-conquer* structure.

## 3. The `ConstructibleSetTools` Module

Our `ConstructibleSetTools` module is a collection of commands for computing with constructible sets. It includes a relatively complete set of basic routines, like building constructible sets, set operations such as difference, intersection and union, and also some advanced functionalities. We illustrate some of them by examples in Sections 3.2, 3.3 and 3.4. In Section 3.1 we discuss software design issues.

### *3.1.* Software design

The design of the `ConstructibleSetTools` module somehow mimics the organization of categories and domains in the computer algebra system `AXIOM` (Jenks and Sutor, 1992). Both regular systems and constructible sets are treated as classes of objects, *regular_system* and *constructible_set* . This follows the implementation strategy of the `RegularChains` library (Lemaire et al., 2006), where a regular chain is a class type *regular_chain*. This implementation technique enhances the extensibility and reusability of our code.

The `RegularChains` library user-interface has been organized into two levels: a set of basic commands for non-expert users and a set of modules with more advanced functionalities. The `ConstructibleSetTools` module lies in this second level.

We also provide flexibility in viewing the data. Since symbolic computation generally involves large expressions, our functions are devised to display their computed results as the types of the objects which they represent, i.e. *regular_chain*, *regular_system* and *constructible_set* . The user can then choose to view more details by our displaying tools such as `Equations` and `Info`.

A special design issue is on the representation and simplification of constructible sets, as discussed in Section 2. One could think of maintaining the objects of type *constructible_set* in a canonical representation. This is the point of view which is followed for most domains in `AXIOM`. For the case of constructible sets, this could imply using decomposition into irreducible components as in (Manubens and Montes, 2006a); the comparative experimentation in (Chen et al., 2007a) suggests that this could be a bottleneck. Instead, we follow a *lazy (but not too lazy) strategy*. In our module, the union of two constructible sets $C_1$ and $C_2$ is simply computed by concatenating the regular system lists encoding $C_1$ and $C_2$. Thus, the representation of the constructible set $C_1 \cup C_2$ may contain some redundant components. However, the command `MakePairwiseDisjoint` can be used to remove them at a relatively low cost. This is achieved by the good computational properties of our representation, presented in Section 2. For computing $C_1 \cap C_2$ and $C_1 \setminus C_2$, however, our code performs algebraic computations such those objects can be represented by regular system lists. This is why we would like to say that our strategy is lazy, but not too lazy.

### *3.2.* Creating constructible sets

The command `GeneralConstruct` provides a synthetic way to create a constructible set. For a given polynomial ring $R$, this command takes as input two lists of polynomials $F$ and $H$, regarded respectively as equations and inequations; optionally it takes also a regular chain $T$. Then, it returns the constructible set equal to $(V(F) \cap W(T)) \setminus V(H)$. In fact, the command `GeneralConstruct` extends the functionalities of the `Triangularize` command in the `RegularChains` library.

**Example 2.** In the `Maple` session below, after loading the `RegularChains` library and the `ConstructibleSetTools` module, we define $R$ to be the polynomial ring $\mathbb{Q}[x \succ y \succ z]$. Note that the second argument of the `PolynomialRing` command specifies the characteristic of the ring.

```
> R := PolynomialRing([x, y, z], 0):
> F := [x*y*z-x*y, y^2-y*z]:
> H := [y]:
> cs := GeneralConstruct(F, H, R);
```

$$cs := constructible\_set$$

```
> lrs := RepresentingRegularSystems(cs, R);
```

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

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

$$[[x, y - z], [z]],\ [[y - 1, z - 1], [1]]$$

Recall that we encode each constructible set by a list of regular systems. The command `RepresentingRegularSystems` gives access to this representation. In addition, the commands `RepresentingChain` and `RepresentingInequations` (not illustrated above) "unwraps" a regular system $[T, h]$ and returns respectively $T$ and $h$. Alternatively, the `Info` command displays directly the polynomials defining a constructible set, a regular system or a regular chain. Therefore, in the above $lrs$, the zero set of the first regular system can be read as $\{(x, y, z) \mid x = 0, y = z, z \neq 0\}$,

and the zero set of the other one is $\{(x, y, z) \mid y = 1, z = 1\}$. The constructible set $cs$ is the union of them.

### *3.3.* Basic operations

**Example 3.** One of the key functionalities is to compute the set theoretical difference of two constructible sets which is also a constructible set.

```
> R := PolynomialRing([x, y, u, v]):
> G := [x^2+y^2-1, u*x-v*y]:
> cs1 := GeneralConstruct(G, [x], R):
> Info(cs1, R);
```

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y, v]]$$

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

```
> cs2 := GeneralConstruct(G, [y], R):
> Info(cs2, R);
```

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y]],\ [[x^2 + y^2 - 1, u, v], [y]]$$

```
> cs3 := Difference(cs1, cs2, R);
```

$$cs3 := constructible\_set$$

```
> Info(cs3, R);
```

$$[[x - 1, y, u], [v]],\ [[x - 1, y, u, v], [1]],$$
$$[[x + 1, y, u], [v]],\ [[x + 1, y, u, v], [1]]$$

**Example 4.** The command `Union` forms the union of two constructible sets, simply by putting all defining regular systems together.

```
> cs4 := Union(cs1, cs2, R); Info(cs4, R);
```

$$cs4 := constructible\_set$$

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y, v]],$$

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

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y]],\ [[x^2 + y^2 - 1, u, v], [y]]$$

Due to our lazy evaluation strategy, the output of `Union` may contain redundancy. For example, the zero set of the regular system

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y, v]]$$

is contained in the zero set of the regular system

$$[[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y]].$$

If an irredundant result is demanded, one can call `MakePairwiseDisjoint` to clean $cs4$.

```
> cs5 := MakePairwiseDisjoint(cs4, R);
```

$$cs5 := constructible\_set$$

```
> Info(cs5, R);
```

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

$$[[x + 1, y, u], [1]],\ [[ux - vy, \left(u^2 + v^2\right) y^2 - u^2], [y]]$$

The constructible set $cs5$ encodes the same set of points as $cs4$, but the zero sets of its defining regular systems are pairwise disjoint.

**Example 5.** The command `RefiningPartition` implements the operation `SMPD` introduced in Section 2.2 and the algorithm it uses is the divide-and-conquer algorithm (Algorithm 2) of Section 2.4. This function can remove the redundancy among a list of constructible sets while preserving geometrical information carried by them.

```
> rp := RefiningPartition([cs1, cs2], R);
```

$$rp := \begin{bmatrix} constructible\_set & [2] \\ constructible\_set & [2,1] \\ constructible\_set & [1] \end{bmatrix}$$

The above output is a matrix in which the first column are constructible sets and the second column are indices showing where the constructible sets come from. In this matrix, the constructible set in the first row with index "2" is the difference $cs2 \setminus cs1$, the second one with index "1, 2" is the intersection $cs1 \cap cs2$ and the third one with index "1" is the difference $cs1 \setminus cs2$. Hence, the three constructible sets in $rp$ form an intersection-free basis of $cs1$ and $cs2$.

### *3.4.* Two advanced operations

This subsection introduces two advanced operations related to constructible sets: *projection* and *rational map image*.

**Example 6.** The `Projection` command is used to project a constructible set of $\overline{\mathbb{K}}^n$ to the space of the first $d$ coordinates for some $1 \leq d < n$. Suppose that we want to answer the following question: what are the conditions on the coefficients of the quadratic equation $x^2 + ax + b = 0$ to have two non-zero solutions, one being the double of the other. Let $x$ and $y$ be the roots of this equation. The calculations below solve our question and the answer is $9b = 2a^2$ provided that $a$ and $b$ are non-zero.

```
> R := PolynomialRing([x, y, a, b]):
> px := x^2+a*x+b:
> py := y^2+a*y+b:
> cs1 := GeneralConstruct([px, py, y-2*x], [x], R):
> cs2 := Projection(cs1, 2, R);
```

$$cs2 := constructible\_set$$

```
> Info(cs2, R);
```

$$[[-9\,b + 2\,a^2], [a, b]]$$

In above example, the second argument "2" of the `Projection` command means that the constructible $cs1$ is projected onto the coordinate space defined by the last two variables.

`RationalMapImage` and `RationalMapPreimage` are two commands for computing respectively the image and preimage of a constructible set under a rational map. As a direct application, we illustrate how to find the implicit representation of a curve from a rational parametrization. The following example shows how to compute the implicit representation of the tacnode curve.

**Example 7.** First, we define two polynomial rings for the source space $S$ and the target space $T$. Note that the default characteristic is zero, hence the second argument of the `PolynomialRing` command can be omitted.

```
> S := PolynomialRing([t]):
> T := PolynomialRing([x, y]):
```

Then we define a rational map $M$ from $S$ to $T$.

```
> mx := (t^3-6*t^2+9*t-2)/(2*t^4-16*t^3+40*t^2-32*t+9):
> my := (t^2-4*t+4)/(2*t^4-16*t^3+40*t^2-32*t+9):
> M := [mx, my];
```

$$M := [\frac{t^3 - 6\,t^2 + 9\,t - 2}{2\,t^4 - 16\,t^3 + 40\,t^2 - 32\,t + 9}, \frac{t^2 - 4\,t + 4}{2\,t^4 - 16\,t^3 + 40\,t^2 - 32\,t + 9}]$$

The image of the whole space under the map $M$ is computed below by the command `RationalMapImage`. We obtain a constructible set $cs$ encoded by three regular systems. Each of the last two is just a point whereas the first one is "almost" a curve.

```
> F := []:
> cs := RationalMapImage(F, M, S, T);
```

$$cs := constructible\_set$$

```
> RepresentingRegularSystems(cs, T);
```

$$[regular\_system, regular\_system, regular\_system]$$

```
> Info(cs, T);
```

$[[2\,x^4 - 3\,yx^2 + y^2 - 2\,y^3 + y^4], [y, \left(-4328\,y^3 + 964\,y^6 - 480\,y^5 - 6858\,y^4 - 888\,y^2 - 2 - 72\,y\right) x^2$
$+892\,y^4 + y + 2104\,y^7 - 88\,y^8 + 32\,y^2 - 2316\,y^6 - 943\,y^5 + 318\,y^3, (10\,y + 2)\,x^2 + 2\,y^3 - y^2 - y]],$
$[[x, y], [1]],\ [[x, y - 1], [1]]$

We denote by $p$ below the equation in the first regular system. It is not hard to prove that the variety of $p$ is the closure of the image $cs$ and therefore is the implicit representation of the tacnode curve. Moreover, the image $cs$ is equal to the variety of $p$ as checked below by the command `IsContained`. This fact implies that the rational parametrization of tacnode curve encodes exactly all the points of it.

```
> p := 2*x^4-3*y*x^2+y^2-2*y^3+y^4:
> cs2 := GeneralConstruct([p], [], T):
> IsContained(cs, cs2, T) and IsContained(cs2, cs, T);
```

$$true$$

## 4. The `ParametricSystemTools` Module

Our `ParametricSystemTools` module is a collection of commands for solving polynomial systems depending on parameters. It is a direct application of the `ConstructibleSetTools` module presented in the previous section. The main commands presented in this section are: `PreComprehensiveTriangularize`, `ComprehensiveTriangularize`, `DiscriminantSet`, `ComplexRootClassification`. These functions can be used to understand the properties of the solution set of a parametric polynomial system $F$, with or without inequations. For instance, one can answer questions like: for which values of the parameters does $F$ have solutions, finitely many solutions, or $N$ solutions for a given $N > 0$ ? We start this section with a few definitions.

Let $F$ be a finite set of polynomials with coefficients in $\mathbb{K}$, parameters in $U = (u_1 \prec \cdots \prec u_d)$, and unknowns $X = (x_1 \prec \ldots \prec x_m)$, that is, $F \subset \mathbb{K}[U, X]$. Recall that $\overline{\mathbb{K}}$ denotes the algebraic closure of $\mathbb{K}$. For each $u \in \overline{\mathbb{K}}^d$, we define $V(F(u)) \subseteq \overline{\mathbb{K}}^m$ as the zero set of $F$ after specializing $U$ at $u$. For a constructible set $cs$ of $\mathbb{K}[U, X]$ and a point $u \in \overline{\mathbb{K}}^d$, we define

$$cs(u) = \{x \in \overline{\mathbb{K}}^m \mid (u, x) \in cs\}.$$

**Specialize well.** Let $T \subset \mathbb{K}[U, X]$ be a regular chain. Let $T_0$ be the set of the polynomials in $T$ involving only the parameters in $U$ and let $T_1$ be the set of the other polynomials in $T$. Let $W \subset \overline{\mathbb{K}}^d$ be the quasi-component of $T_0$ regarded as a regular chain in $\mathbb{K}[U]$. The regular chain $T$ *specializes well* at a point $u$ of $W$ if $T_1(u) \subset \overline{\mathbb{K}}[X]$ is a regular chain after specialization and no initial of polynomials in $T_1$ vanishes during the specialization. The regular system $rs = [T, h]$ *specializes well* at a point $u$ of $W$ if $T$ specializes well at $u$ and $h(u)$ is regular w.r.t $\mathrm{sat}(T_1(u))$.

### *4.1.* Pre-comprehensive triangular decomposition

A *pre-comprehensive triangular decomposition* (PCTD) of a constructible set $cs \subset \mathbb{K}[U, X]$ is a family of regular systems $\mathcal{R}$ satisfying the following property: for each $u \in \overline{\mathbb{K}}^d$, denoting by $\mathcal{R}_u$ the subfamily of all regular systems in $\mathcal{R}$ that specialize well at $u$, we have

$$cs(u) = \bigcup_{rs \in \mathcal{R}_u} Z(rs(u)).$$

In the `Maple` session below, $F$ and $H$ are two lists of polynomials in $\mathbb{Q}[x, y, s]$ regarded as equations and inequations, respectively. The constructible set $cs$, defined by $F$ and $H$, is represented by two regular systems. Observe that, for $s = 0$, the regular system $[[(y + 1)x - s, y^2 - s + y], -1 + 2\,s - y]$, specializes to $[[(y + 1)x, y^2 + y], -1 - y]$, which is not a regular system since $y + 1$, the initial of the polynomial $(y + 1)x$, is a zero-divisor modulo $y^2 + y$. These "bad specializations" are discovered by the command `PreComprehensiveTriangularize`. This is why the call `PreComprehensiveTriangularize(F,H,1,R)` returns four regular systems instead of two. (Note that the third parameter "1" specifies that the least variable of the polynomial ring $R$ is regarded as a parameter.) The third and fourth regular systems in *pctd* correspond to the "bad parameter values" $s = 0$ and $-3 + 4s = 0$ of the first regular system of $cs$.

**Example 8** (PCTD)**.**

```
> R := PolynomialRing([x,y,s]):
> F := [s-(y+1)*x,s-(x+1)*y]:
> H := [x+y-1]:
> cs := GeneralConstruct(F,H,R):
> dec := RepresentingRegularSystems(cs, R);
> map(Info,dec,R);
```

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

$$[[[(y+1)\,x - s, y^2 - s + y], [-1 + 2\,s - y]], [[x + 1, y + 1, s], [1]]]$$

```
> pctd := PreComprehensiveTriangularize(F, H, 1, R);
> map(Info, pctd, R);
```

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

$$[[[(y+1)\,x - s, y^2 - s + y], [-1 + 2\,s - y]], [[x + 1, y + 1, s], [1]],$$
$$[[x, y, s], [1]], [[2\,x + 3, 2\,y + 3, -3 + 4\,s], [1]]]$$

**Remark 1.** We would like to point out here how the `Projection` command is implemented. Recall that the function call `Projection(cs, d, R)` computes the projection image of a constructible set $cs$ on the parameter space defined by the last $d$ variables of the polynomial ring $R$. The trick is to first compute a pre-comprehensive triangular decomposition *pctd* of $cs$ regarding the last $d$ variables as parameters, and then to compute the union of the sets $D(rs)$ for all $rs$ in *pctd*, where $D(rs)$ is the set of the parameter values $u$ at which $rs$ specializes well.

### *4.2.* Comprehensive triangular decomposition

Comparing with the notion of pre-comprehensive triangular decomposition, that of *comprehensive triangular decomposition (CTD)* provides additional information on the geometry of the input polynomial system. A CTD of a constructible set $cs$ in $\mathbb{K}[U, X]$ comprises: (1) a finite partition of the parameter space into cells, each of them given by a constructible set, and (2) for each cell $C$, a family $\mathcal{R}_C$ of regular systems which specialize well at any point of $C$ and such that, for all $u \in C$ we have:

$$cs(u) = \bigcup_{rs \in \mathcal{R}_C} Z(rs(u)).$$

The following example shows how to compute the comprehensive triangular decomposition of a constructible set by `ComprehensiveTriangularize` command. Two ways are available. First the list $F$ of equations and the list $H$ of inequations are passed to `ComprehensiveTriangularize`. Secondly, we build the constructible set $cs$ from $F$ and $H$; then we pass it to `ComprehensiveTriangularize`.

**Example 9** (CTD)**.**

```
> R := PolynomialRing([x, y, s]):
> F := [s-(y+1)*x, s-(x+1)*y]:
> H := [x+y-1]:
> ctd := ComprehensiveTriangularize(F, H, 1, R);
```

$$ctd := [regular\_system, regular\_system, regular\_system, regular\_system],$$
$$[[constructible\_set, [1]], [constructible\_set, [2, 3]], [constructible\_set, [4]]]$$

```
> cs := GeneralConstruct(F, H, R);
> ctd := ComprehensiveTriangularize(cs, 1, R);
```

$$ctd := [regular\_system, regular\_system, regular\_system, regular\_system],$$
$$[[constructible\_set, [1]], [constructible\_set, [2, 3]], [constructible\_set, [4]]]$$

```
> cs1 := ctd[2][1][1]: Info(cs1, R);
> cs2 := ctd[2][2][1]: Info(cs2, R);
> cs3 := ctd[2][3][1]: Info(cs3,R);
```

$$[[], [-3 + 4\,s, s]]$$
$$[[s], [1]]$$
$$[[-3 + 4\,s], [1]]$$

The number "1" in the command `ComprehensiveTriangularize` means that the least variable is the parameter. The output of `ComprehensiveTriangularize` consists of two parts: the first one is a list of regular systems; the second part is a list of indexed constructible sets. Each of these constructible sets encodes a cell in the parameter space above which the solutions of the input system are given by the regular systems whose indices are in the list associated with the constructible set. For our particular example, we see that, if $(-3 + 4s)s \neq 0$ holds, then the solutions of the input system are those of the first regular system. If $s = 0$, the solutions of the input system are given by the the union of those of the second and third regular systems. Finally, if $(-3+4s) = 0$, the solutions of the input system are that of the fourth regular system.

### *4.3.* Complex root classification

The `ParametricSystemTools` module provides two commands for investigating the number of solutions of a polynomial system depending on parameters. The input of these two commands can be either a polynomial system with or without inequations or, a constructible set.

**Example 10.** The function `DiscriminantSet` is used for determining the *discriminant set* of a constructible set $cs$, which is defined as the set of all points $u \in \overline{\mathbb{K}}^d$ for which $cs(u)$ is empty or infinite. In this example, the argument "2" means we see the last two variables as parameters.

```
> R := PolynomialRing([x, y, u, v]):
> F := [x^2+y^2-1, u*x-v*y]:
> cs := DiscriminantSet(F, [x], 2, R);
> Info(cs, R);
```

$$cs := constructible\_set$$

$$[[u, v], [1]],\ [[u^2 + v^2], [v]],\ [[v], [u]]$$

**Example 11.** The function `ComplexRootClassification` is used to compute the number of distinct complex roots of a parametric polynomial system depending on parameters. In this example, the argument "1" means we see the last variable as the parameter.

```
> R := PolynomialRing([x, y, s]):
> F := [s-(y+1)*x, s-(x+1)*y]:
> H := [x+y-1]:
> crc := ComplexRootClassification(F, H, 1, R);
```

$$crc := [[constructible\_set, 1], [constructible\_set, 2]]$$

The output is a list of pairs, which shows all the possible numbers of complex roots together with the corresponding conditions on the parameter. Thus, this output provides a complete complex root classification of the input system.

```
> map(x->[Info(x[1], R), x[2]], crc):
```

$$[[[4\,s + 1], [1]], [[-3 + 4\,s], [1]], 1], [[[s], [1]], [[], [s, -3 + 4\,s, 4\,s + 1]], 2]]$$

For this example, the output can be read as follows: If $(4s+1)(-3+4s) = 0$, then the system has 1 complex root. Otherwise, the system has 2 complex roots.

### *4.4.* A small comparison with `SACGB`

We report here a brief comparison of our function `ComprehensiveTriangularize` with the `Maple` implementation `SACGB` on comprehensive Gröbner basis by Suzuki and Sato (2006). The comparison between `ComprehensiveTriangularize`, the RegSer function of `Epsilon` by Wang (2000) and the function DISPGB of `DPGB` by Montes (2002) can be found in Chen et al. (2007a).

Table 2 illustrates the timing and the length of the output regarded as a string on the 6 examples from Suzuki and Sato (2006). The tests are performed in `Maple` 9.5 on an Intel Pentium 4 machine (2.60GHz CPU, 1.0GB memory). Here we list the defining polynomials of

| | `SACGB` | | | `CTD` | | |
|---|---|---|---|---|---|---|
| Sys | Time(s) | # Segs | Length | Time(s) | # Cells | Length |
| 1 | 38.1 | 13 | 120788 | 5.8 | 4 | 591 |
| 2 | Error | – | – | $>$ 1 hour | – | – |
| 3 | $>$ 1 hour | – | – | $>$ 1 hour | – | – |
| 4 | 904.7 | 27 | 23398 | 2.0 | 9 | 958 |
| 5 | $>$ 1 hour | – | – | 1.8 | 7 | 961 |
| 6 | $>$ 1 hour | – | – | 1.3 | 5 | 864 |

**Table 2** Comparison with `SACGB` on 6 examples

cells (segments) on the parameter space for the first example in Table 2. The output by `SACGB`

consists of the following 13 segments:

$$C_1 : b = 0, a \neq 0$$

$$C_2 : b = 0, a^4 = 0, a \neq 0$$

$$C_3 : b^3 = 0, a = 0, b \neq 0$$

$$C_4 : 729a^4 + 64b^3 = 0, ab \neq 0$$

$$C_5 : b = 0, a^8 = 0, a \neq 0$$

$$C_6 : b^6 = 0, a = 0, b \neq 0$$

$$C_7 : 16767a^4 + 5632b^3 = 0, b^6 = 0, ab \neq 0$$

$$C_8 : a = 0, b \neq 0$$

$$C_9 : b = 0, a = 0$$

$$C_{10} : ab(16767a^4 + 5632b^3)(-4096b^3 + 729a^4)(729a^4 + 64b^3)^2 \neq 0$$

$$C_{11} : -4096b^3 + 729a^4 = 0, ab \neq 0$$

$$C_{12} : ab(16767a^4 + 5632b^3)(-4096b^3 + 729a^4)(729a^4 + 64b^3) \neq 0,$$
$$-2939328b^3a^4 - 262144b^6 + 531441a^8 = 0$$

$$C_{13} : 16767a^4 + 5632b^3 = 0, ab \neq 0.$$

The partition obtained by `ComprehensiveTriangularize` contains 4 nonempty cells, listed below:

$$D_1 : 729a^4 + 64b^3 = 0, b \neq 0, a \neq 0$$

$$D_2 : 729a^4 + 64b^3 \neq 0, 729a^4 - 4096b^3 \neq 0, b \neq 0$$
$$\text{or } b = 0, a \neq 0$$

$$D_3 : 729a^4 - 4096b^3 = 0, b \neq 0, a \neq 0$$

$$D_4 : a = 0, b = 0.$$

Note that there exist inconsistent segments in `SACGB`'s output and these segments may have common part, while all $D_i$'s are nonempty and pairwise disjoint.

There are a few other packages or programs available related to solving parametric systems: `DPGB`, `DV`, `RootFinding[Parametric]`, `SACGB`, `DISCOVERER`, `Epsilon`, `WSolve`, etc. The first five packages are based on Gröbner bases approach while the rest three are implemented via triangular decompositions. Among the packages based on triangular decompositions, `DISCOVERER` focuses on real solving; `WSolve` does not have dedicated functions for parametric system solving; `Epsilon` can be applied to parametric polynomial system solving, but it usually computes more than needed since the parameters are not prescribed. Our `ParametricSystemTools` module explicitly distinguishes variables and parameters and is a tool particularly aiming at parametric polynomial system solving.

## 5. Solver Verification

Symbolic solvers are highly complex software. They implement sophisticated algorithms, which are generally at the level of on-going research. Moreover, in most computer algebra systems, the `solve` command involves nearly the entire set of libraries in the system, challenging the most advanced operations on matrices, polynomials, algebraic and modular numbers, polynomial ideals, etc.

Given a polynomial system $F$ and a set of components $C_1, \ldots, C_e$, it is hard, in general, to tell whether the union of $C_1, \ldots, C_e$ corresponds exactly to the solution set $V(F)$ or not. Actually, solving this verification problem is generally (at least) as hard as solving the system $F$ itself.

Because of the high complexity of symbolic solvers, developing verification algorithms and reliable verification software tools is a clear need. However, this verification problem has received little attention in the literature. In Chen et al. (2007b) we proposed a procedure in order to verify polynomial system solvers based on triangular decompositions.

In this section, we consider the verification of software packages computing polynomial GCDs modulo regular chains. Below, we review this GCD notion, see (Moreno Maza, 1999).

Let $T \in \mathbb{K}[X]$ be a regular chain and let $f_1, f_2 \in \mathbb{K}[X]$ be non-constant polynomials with the same main variable $v$. We assume that $v$ is larger than any variable occurring in $T$ and that the initials of $f_1, f_2$ are regular w.r.t. the saturated ideal of $T$. A non-zero polynomial $g \in \mathbb{K}[X]$ is a *polynomial GCD modulo* $T$ of $f_1, f_2$ if the following conditions hold:

($i$) the leading coefficient of $g$ w.r.t. $v$ is regular w.r.t. $\mathrm{sat}(T)$,

($ii$) if $\deg(g, v) > 0$ then $f_1$ and $f_2$ belong to $\mathrm{sat}(T \cup \{g\})$,

($iii$) there exist $a_1, a_2 \in \mathbb{K}[X]$ such that $g \equiv a_1 f_1 + a_2 f_2 \mod \mathrm{sat}(T)$.

Verifying this third condition is non-trivial and can be achieved via Gröbner basis computations.

Let us suppose now that we have at hand two implementations computing polynomial GCDs modulo regular chains. Assume first that on input $f_1, f_2, T$ they return respectively two different polynomials $g$ and $g'$ whose initials are regular w.r.t $\mathrm{sat}(T)$. Let us assume that the first implementation is trusted, hence $g$ is correct. To check that $g'$ is correct it is sufficient to check that the saturated ideals $\mathrm{sat}(T \cup \{g\})$ and $\mathrm{sat}(T \cup \{g'\})$ are equal, which can be done simply by pseudo-division.

The difficulties start when one implementation splits the computation and the other does not, or even worse, when they both split the computations but use different decompositions of $\mathrm{sat}(T)$. To highlight this point, let us assume that on input $f_1, f_2, T$ the trusted implementation returned $g$ whereas the other implementation splits $T$ into $T_1, T_2$ (this means that $\mathrm{sat}(T) = \mathrm{sat}(T_1) \cap \mathrm{sat}(T_2)$ holds) and that modulo $T_i$ the returned polynomial GCD is $g_i$, for $i = 1, 2$. Observe that if we have

$$W(T \cup \{g\}) \quad = \quad W(T_1 \cup \{g_1\}) \cup W(T_2 \cup \{g_2\}) \tag{1}$$

then we also have

$$\sqrt{\mathrm{sat}(T \cup \{g\})} \quad = \quad \sqrt{\mathrm{sat}(T_1 \cup \{g_1\})} \cap \sqrt{\mathrm{sat}(T_2 \cup \{g_2\})}. \tag{2}$$

Indeed, the Zariski closure of the quasi-component $W(T)$ (for any regular chain $T$) equals $V(\mathrm{sat}(T))$. If the saturated ideals of the regular chains $T \cup \{g\}$, $T_1 \cup \{g_1\}$ and $T_2 \cup \{g_2\}$ are radical (which is easy to check in characteristic zero or in dimension zero) we obtain the property that we want to check, that is:

$$\mathrm{sat}(T \cup \{g\}) \quad = \quad \mathrm{sat}(T_1 \cup \{g_1\}) \cap \mathrm{sat}(T_2 \cup \{g_2\}) \tag{3}$$

Equation (1) is an equality between constructible sets (which may be not algebraic varieties). Checking that this equality holds can be done by means of our `ConstructibleSetTools` module. This computation is relatively easy with the algorithms of Section 2.

Equation (3) is an equality between ideals. Checking it can be done via Gröbner basis computations but this is potentially expensive (due to the computation of bases for saturated ideals).

Observe that Equation (3) might hold while Equation (1) does not. However, during our experimentation we have never encountered such case. We have used this verification procedure for the fast GCD algorithms reported in Li et al. (2008) comparing them against the trusted implementation of the `RegularChains` library.

## 6. Related Work

This section is a survey of algorithms and implementations related to constructible sets. Recall that a constructible set is a finite union of subsets $W = A \setminus B$ with $A$ and $B$ being varieties. Different representations for $W$ give quite different methods to realize basic operations like difference, intersection, and projection.

**Gröbner basis approach** The above two varieties $A$ and $B$ can be given by two reduced Gröbner bases, as in O'Halloran and Schilmoeller (2002), Kemper (2007), Schauenburg (2007), Manubens and Montes (2006b), etc. In Manubens and Montes (2006b), as part of computing canonical comprehensive Gröbner systems, the authors proposed an algorithm to describe the segments (constructible sets) in a canonical way, where a constructible set is represented by a P-tree (each node, except the root, is a prime ideal) with some additional properties.

The problem of computing the difference or intersection of two constructible sets boils down to manipulating polynomial ideals like computing the intersection of two ideals. In Sit (1998), the author proposed algorithms for testing inclusion and equality relations between two constructible sets. The main technique is based on radical membership tests with Gröbner bases. Similar technique was also used in Chen et al. (2004) to make the constructible subsets of parameter space consistent. The projection of a variety may be seen as a refinement of the classical elimination and extension theorems (Cox et al., 1992). In Schauenburg (2007), the authors describe an algorithm to compute the projection image of a variety. This approach is geometrical and there exists a certain analogy with the one based on the notion of well-specialization in (Chen et al., 2007a).

**Triangular set approach** By the technique of triangular decompositions as introduced in (Wu, 1984), the set $W = A \setminus B$ can be decomposed into a union of zero sets of triangular systems (Wang, 2001; Chen and Wang, 2002). Each triangular system is a pair $[T, h]$, where $T$ is a triangular set and $h$ is a polynomial. The set of points encoded by the pair $[T, h]$ is $V(T) \setminus V(h)$. A very first problem is that this set may be empty, and there are several constraints added to a triangular system.

In (Chou and Gao, 1992; Gao and Wang, 2003), triangular sets are normalized, that is, the initials only involve parameters. In (Wang, 2000, 2001), Wang proposed a notion of "regular system" (stronger than the one used in Section 2) in which both the triangular sets and the inequations are normalized. Unfortunately, as shown by the complexity result of (Dahan and Schost, 2004), "normalizing" regular chains tends to blow up coeffcients dramatically. In (Chen et al., 2007a) the authors defined a weaker notion of a regular system (the one of Section 2) where normalization is not involved. Since the zero set of a regular system is always nonempty (in fact unmixed), a constructible set is empty if and only if it is given by an empty list of regular systems.

The difference of two constructible sets can be computed naively, that is, by reducing the problem to compute the union or intersection of two varieties. In (Chen et al., 2007a), an efficient algorithm was developed for computing the difference of the zero sets of two regular systems. The basic idea there is to make better use of the triangular structure of the input and a similar idea applies to compute the intersection. A sketch of this algorithm is given in Section 2.1.

To compute the projection of a constructible set, one may first decompose it into the union of zero sets of triangular systems. Then it is enough to compute the projection of the zero set of each triangular system. Afterwards, one can apply pseudo-division to eliminate variables of a triangular system one by one. This approach was proposed by Wu (1990) and further developed in (Chen and Wang, 2002; Wang, 2001; Chou and Gao, 1992). There is an interesting result stated in (Wang, 2001, 2005), which pointed out that the projection of the zero set of a regular system (in the sense of Wang) is simply the common zeros of some polynomials in the regular system. This property is called the *strong projection property* of a (Wang) regular system.

The packages mentioned earlier on parametric polynomial system are also related to constructing and manipulating constructible sets. There is another related package, `QuasiAlgebraicSet`, by Sit (1998) implemented in `AXIOM`. Although there are many related implementations, to our knowledge, there is no package which dedicates to manipulating constructible sets in a systematic way before the development of our `ConstructibleSetTools` module.

## References


Aubry, P., Lazard, D., Moreno Maza, M., 1999. On the theories of triangular sets. J. Symb. Comp. 28 (1-2), 105–124.

Bach, E., Driscoll, J. R., Shallit, J., 1990. Factor refinement. In: Proc. SODA. pp. 201–211.

Buchberger, B., 1965. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, University of Innsbruck.

Chen, C., Golubitsky, O., Lemaire, F., Moreno Maza, M., Pan, W., 2007a. Comprehensive Triangular Decomposition. Vol. 4770 of LNCS. Springer Verlag, pp. 73–101.

Chen, C., Li, L., Moreno Maza, M., Pan, W., Xie, Y., 2008. Computations of intersection-free bases. Tech. rep., The University of Western Ontario.

Chen, C., Moreno Maza, M., Pan, W., Xie, Y., 2007b. On the verification of polynomial system solvers. In: Proceedings of AWFS 2007. pp. 116–144.

Chen, X., Li, P., Lin, L., Wang, D., 2004. Proving geometric theorems by partitioned-parametric Gröbner bases. In: Automated Deduction in Geometry. pp. 34–43.

Chen, X., Wang, D., 2002. The projection of quasi variety and its application on geometric theorem proving and formula deduction. In: ADG. pp. 21–30.

Chou, S., Gao, X., 1992. Solving parametric algebraic systems. In: Proc. ISSAC'92. pp. 335–341.

Cox, D., Little, J., O'Shea, D., 1992. Ideals, Varieties, and Algorithms, 1st Edition. Spinger-Verlag.

Dahan, X., Moreno Maza, M., Schost, É., Xie, Y., 2006. On the complexity of the D5 principle. In: Proc. of *Transgressive Computing 2006*. Granada, Spain.

Dahan, X., Schost, É., 2004. Sharp estimates for triangular sets. In: ISSAC 04. ACM, pp. 103–110.

Della Dora, J., Dicrescenzo, C., Duval, D., 1985. About a new method for computing in algebraic number fields. In: Proc. EUROCAL 85 Vol. 2. Vol. 204 of Lect. Notes in Comp. Sci. Springer-Verlag, pp. 289–290.

Gao, X., Wang, D., 2003. Zero decomposition theorems for counting the number of solutions for parametric equation systems. In: Proc. ASCM 2003. pp. 129–144.

Hong, H., 1992. Simple solution formula construction in cylindrical algebraic decomposition based quantifier elimination. In: Proc. ISSAC. pp. 177–188.

Jenks, R. D., Sutor, R. S., 1992. AXIOM, The Scientific Computation System. Springer-Verlag, AXIOM is a trade mark of NAG Ltd, Oxford UK.

Kalkbrener, M., 1993. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. J. Symb. Comp. 15, 143–167.

Kemper, G., 2007. Morphisms and constructible sets: Making two theorems of chevalley constructive, preprint.

Kogan, I. A., Moreno Maza, M., Jul. 2002. Computation of canonical forms for ternary cubics. In: Mora, T. (Ed.), Proc. ISSAC 2002. ACM Press, pp. 151–160.

Lemaire, F., Moreno Maza, M., Xie, Y., 2005. The `RegularChains` library. In: Ilias S. Kotsireas (Ed.), *Maple Conference 2005*. pp. 355–368.

Lemaire, F., Moreno Maza, M., Xie, Y., 2006. Making a sophisticated symbolic solver available to different communities of users. In: Proc. of *Asian Technology Conference in Mathematics*'06.

Li, X., Moreno Maza, M., Rasheed, R., Schost, E., 2008. The modpn library: bringing fast polynomial arithmetic into maple. In: Proc. Milestones in Computer Algebra. pp. 72–60.

Manubens, M., Montes, A., 2006a. Improving the dispgb algorithm using the discriminant ideal. J. Symb. Comput. 41 (11), 1245–1263.

Manubens, M., Montes, A., 2006b. Minimal canonical comprehensive Gröbner system. ArXiv:math.AC/0611948.

Montes, A., 2002. A new algorithm for discussing gröbner bases with parameters. J. Symb. Comput. 33 (2), 183–208.

Moreno Maza, M., 1999. On triangular decompositions of algebraic varieties. Tech. Rep. TR 4/99, NAG Ltd, Oxford, UK, http://www.csd.uwo.ca/∼moreno.

Moreno Maza, M., Rioboo, R., 1995. Polynomial gcd computations over towers of algebraic extensions. In: Proc. AAECC-11. Springer, pp. 365–382.

O'Halloran, J., Schilmoeller, M., 2002. Gröbner bases for constructible sets. Journal of Communications in Algebra 30 (11).

Schauenburg, P., 2007. A Gröbner-based treatment of elimination theory for affine varieties. JSC 42 (9), 859–870.

Sit, W., 1998. Computations on quasi-algebraic sets. In: Liska, R. (Ed.), Electronic Proceedings of IMACS ACA'98.

Suzuki, A., Sato, Y., 2006. A simple algorithm to compute comprehensive Gröbner bases using Gröbner bases. In: ISSAC. pp. 326–331.

Wang, D., 2000. Computing triangular systems and regular systems. J. Sym. Comp. 30 (2), 221–236.

Wang, D., 2001. Elimination Methods. Springer.

Wang, D., 2005. The projection property of regular systems and its application to solving parametric polynomial systems. In: Dolzmann, A., Seidl, A., Sturm, T. (Eds.), Algorithmic Algebra and Logic. Herstellung und Verlag, Norderstedt, pp. 269–274.

Wu, W. T., 1984. Basic principles of mechanical theorem proving in elementary geometries. J. Sys. Sci. and Math. Scis 4, 207–235.

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

Wu, W. T., 1990. On a projection theorem of quasi-varieties in elimination theory. Chinese Ann. Math. Ser. B. (11), 220–226.

Yang, L., Zhang, J., 1991. Searching dependency between algebraic equations: an algorithm applied to automated reasoning. Tech. Rep. IC/89/263, International Atomic Energy Agency, Miramare, Trieste, Italy.