Front. Comput. Sci. China 2008, 2(1): 55–66
DOI 10.1007/s11704-008-0006-y

RESEARCH ARTICLE

On the verification of polynomial system solvers

Changbo CHEN, Marc MORENO MAZA (✉), Wei PAN, Yuzhen XIE

The University of Western Ontario, London ON N6A 5B7, Canada

© Higher Education Press and Springer-Verlag 2008

Abstract We discuss the verification of mathematical software solving polynomial systems symbolically by way of triangular decomposition. Standard verification techniques are highly resource consuming and apply only to polynomial systems which are easy to solve. We exhibit a new approach which manipulates constructible sets represented by regular systems. We provide comparative benchmarks of different verification procedures applied to four solvers on a large set of well-known polynomial systems. Our experimental results illustrate the high efficiency of our new approach. In particular, we are able to verify triangular decompositions of polynomial systems which are not easy to solve.

Keywords software verification, polynomial system solver, triangular decomposition

1 Introduction

Solving systems of non-linear, algebraic or differential equations, is a fundamental problem in mathematical science. It has been studied for centuries and has stimulated many research developments. Algorithmic solutions can be classified into three categories: numeric, symbolic and hybrid numeric-symbolic. The choice for one of them depends on the characteristics of the system of equations to solve. For instance, it depends on whether the coefficients are known exactly or are approximations obtained from experimental measurements. This choice depends also on the expected answers, which could be a complete description of all the solutions, or only the real solutions, or just one sample solution among all of them.

Symbolic solvers are powerful tools in scientific computing: they are well suited for problems where the desired output must be exact and they have been applied successfully in areas like digital signal processing, robotics, theoretical physics, cryptology, dynamical systems, with many important outcomes. Reference [1] gives an overview of these applications.

Received September 4, 2007; accepted December 10, 2007

E-mail: moreno@csd.uwo.ca

Symbolic solvers are also highly complex software. Firstly, 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.

Secondly, algorithms for solving systems of polynomial equations are by nature of exponential-space complexity. Consequently, symbolic solvers are extremely time-consuming when applied to large examples. Even worse, intermediate expressions can grow to enormous size and may halt the computations, even if the result is of moderate size. The implementation of symbolic solvers, then, requires techniques that go far beyond the manipulation of algebraic or differential equations, such as efficient memory management, data compression, parallel and distributed computing, etc.

Last, but not least, the precise output specifications of a symbolic solver can be quite involved. Indeed, given an input polynomial system F, defining what a symbolic solver should return implies describing what the geometry of the solution set $\mathbf{V}(F)$ of F can be. For an arbitrary F, the set $\mathbf{V}(F)$ may consist of components of different natures and sizes: points, lines, curves, surfaces. This leads to the following difficult challenge.

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 $\mathbf{V}(F)$ 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 clearly a need. However, this verification problem has received little attention in the literature. In this paper, we present new techniques for verifying a large class of symbolic solvers. We also report on intensive experimentation illustrating the high efficiency of our approach w.r.t. known techniques.

We assume that each component of the solution set $\mathbf{V}(F)$ is given by a so-called regular system. This is a natural assumption in symbolic computations, well-developed in the literature under different terminologies in Refs. [2,3] and the references therein. In broad words, a regular system consists of several polynomial equations with a triangular shape

$$p_1(x_1)=p_2(x_1,x_2)=\cdots=p_i(x_1,x_2,\ldots,x_n)=0,$$

and a polynomial inequality

$$h(x_1,\ldots,x_n)\neq 0,$$

such that there exists (at least) one point $(a_1,\ldots,a_n)$ satisfying the above equations and inequality. Note that these polynomials may contain parameters.

Let us consider the following well-known system F taken from Ref. [4].

$$\begin{cases} x^{31}-x^6-x-y &= 0\\ x^8-z &= 0\,.\\ x^{10}-t &= 0 \end{cases}$$

We aim at solving this system for $x>y>z>t$, that is, expressing x as a function of y, z, t, then y as a function of z, t and z as a function of t. One possible decomposition is given by the three regular systems below:

$$\begin{cases} (t^4-t)\,x-ty-z^2 &= 0\\ t^3y^2+2t^2z^2y+\left(-t^6+2t^3+t-1\right)z^4 &= 0\\ z^5-t^4 &= 0\\ t^4-t &\neq 0 \end{cases},$$

$$\begin{cases} x^2-z^4 &= 0\\ y+t^2z^2 &= 0\\ z^5-t &= 0\\ t^3-1 &= 0 \end{cases}, \qquad \begin{cases} x &= 0\\ y &= 0\\ z &= 0\\ t &= 0 \end{cases}.$$

Another decomposition is given by these other three regular systems:

$$\begin{cases} (t^4-t)\,x-ty-z^2 &= 0\\ tzy^2+2z^3y-t^8+2t^5+t^3-t^2 &= 0\\ z^5-t^4 &= 0\\ z(t^4-t) &\neq 0 \end{cases},$$

$$\begin{cases} zx^2-t &= 0\\ ty+z^2 &= 0\\ z^5-t &= 0,\\ t^3-1 &= 0\\ tz &\neq 0 \end{cases} \qquad \begin{cases} x &= 0\\ y &= 0\\ z &= 0\\ t &= 0 \end{cases}.$$

These two decompositions look slightly different (in particular, the second components) and one could think that, if each of them was produced by a different solver, then at least one of these solvers has a bug. In fact, both decompositions are valid, but proving that they encode the solution set $\mathbf{V}(F)$ is not feasible without computer assistance. However, proving that they define the same set of points can be achieved by an expert hand without computer assistance. This is an important observation that will guide us in this work.

Let us consider now an arbitrary input system F and a set of components $C_1,\ldots,C_e$ encoded by regular systems $S_1,\ldots,S_e$ respectively. The usual approach for verifying that $C_1,\ldots,C_e$ correspond exactly to the solution set $\mathbf{V}(F)$ is as follows.

1) First, one checks that each candidate component C_i is actually contained in $\mathbf{V}(F)$. This essentially reduces to substitute the coordinates of the points given by C_i into the polynomials of F: if all these polynomials vanish at these points, then C_i is a component of $\mathbf{V}(F)$; otherwise, (and up to technical details that we will skip in this introduction) C_i is not a component of $\mathbf{V}(F)$.

2) Second, one checks that $\mathbf{V}(F)$ is contained in the union of the candidate components $C_1,\ldots,C_e$ by:

(2.1) computing a polynomial system G such that $\mathbf{V}(G)$ corresponds exactly to $C_1,\ldots,C_e$,

(2.2) checking that every solution of $\mathbf{V}(F)$ cancels the polynomials of G.

Steps (2.1) and (2.2) can be performed using standard techniques based on computations of Gröbner bases, as we discuss in Section 6.1. These calculations are very expensive, as shown by our experimentation, reported in Section 7.

In this paper, we propose a different approach, summarized in non-technical language in Section 2. The main idea is as follows. Instead of comparing a set of candidate components $C_1,\ldots,C_e$ against the input system F, we compare this set against the output $D_1,\ldots,D_f$ produced by another solver. Both this solver and the comparison process are assumed to be validated. Hence, the candidate set of components $C_1,\ldots,C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ if and only if the comparison process shows that $D_1,\ldots,D_f$ and $C_1,\ldots,C_e$ define the same solution set.

The technical details of this new approach are given in Sections 3–6. In Section 3, we review the fundamental algebraic concepts and operations involved in our work. In particular, we specify the kind of solvers that we consider in this study, namely those solving polynomial systems by means of triangular decompositions in the so-called sense of Lazard. (A precise definition is given in Section 3.) This choice is motivated by the following reasons. First, the case of decompositions in the sense of Kalkbrener was treated in Ref. [2], via Gröbner basis computations. Second, most algorithms computing triangular decompositions use the sense of Lazard and no verification tool for those has been reported prior to our work. We leave for future research the verification of Kalkbrener's decompositions by means of more efficient techniques than those reported in Ref. [2].

The key concept behind triangular decomposition in the sense of Lazard is that of a constructible set, so we dedicate Section 4 to it. Section 5 is a formal and complete presentation of our process for comparing triangular decompositions. In Section 6, we summarize the different verification procedures that are available for triangular decompositions, including our new approach. In Section 7, we report on experimentation with these verification procedures. Our data illustrate the high efficiency of our new approach.

2 Methodology

Let us consider again an arbitrary input polynomial system F and a set of components $C_1,\ldots,C_e$ encoded by regular systems $S_1,\ldots,S_e$, respectively. As mentioned in the introduction, checking whether $C_1,\ldots,C_e$ corresponds exactly to the solution set $\mathbf{V}(F)$ of F can be done by means of Gröbner bases computations. This verification process is quite simple, see Section 6, and its implementation is straightforward, Thus, if the underlying Gröbner basis engine is reliable, such verification tool can be regarded as safe. Reference [5] relies on a similar assumption.

Unfortunately, this verification process is highly expensive. Even worse, as shown by our experimental results in Section 7, this verification process is unable to check many triangular decompositions that are easy to compute.

We propose a new approach in order to overcome this limitation. We assume that we have at hand a reliable solver computing triangular decompositions of polynomial systems. We believe that this reliability can be acquired over time by combining several features.

- Checking the solver with a verification tool based on Gröbner bases for input systems of moderate difficulty.
- Using the solver for input systems of higher difficulty where the output can be verified by theoretical arguments, an example of such input system is given in Ref. [6].
- Involving the library supporting the solver in other applications.
- Making the solver widely available to potential users.

Suppose that we are currently developing a new solver computing triangular decompositions. In order to verify the output of this new solver, we can take advantage of the reliable solver.

This may sound natural and easy in the first place, but this is actually a wrong impression. Indeed, as shown in the introduction, two different solvers can produce two different, but valid, triangular decompositions for the same input system. Checking that these two triangular decompositions encode the same solution set boils down to compute the differences of two constructible sets. This is a non-trivial operation, and the survey paper [7] gives the details.

The first contribution of our work is to provide a relatively simple, but efficient, procedure for computing the set theoretical differences between two constructible sets in Section 5. Such procedure can be used to develop a verification tool for our new solver by means of our reliable solver. Moreover, this procedure is sufficiently straightforward to implement such that it can be trusted after a relatively short period of testing, as the case for the verification tool based on Gröbner bases computations.

The second contribution of our work is to illustrate the high efficiency of this new verification tool. In Section 7, we consider four solvers computing triangular decomposition of polynomial systems:

- the command Triangularize of the RegularChains library [8] in MAPLE,
- the TRIADE solver of the BasicMath library [9] in ALDOR,
- the commands RegSer and SimSer of the Epsilon library [10] in MAPLE.

We have run these four solvers on a large set of well-known input systems taken from Refs. [11–13]. For those systems for which this is feasible, we have verified their computed triangular decompositions with a verification tool based on Gröbner bases computations. Then, for each input system, we have compared all its computed triangular decompositions by means of our new verification tool.

Based on our experimentation data reported in Section 7, we make the following observations.

- All computed triangular decompositions, that could be checked via Gröbner bases computations, are correct.
- However, the verification tool based on Gröbner bases computations failed to check many examples by running out of computer memory.
- For each input system F, all pairs of triangular decompositions of F could be compared successfully by our new verification tool.
- Moreover, for any system F to which all verification tools could be applied, our new approach runs much faster.

This suggests that the four solvers and our new verification tool have a good level of reliability. Moreover, it allows to process cases that were previously out of reach.

3 Preliminaries

Let $\mathbb{K}[Y] := \mathbb{K}[Y_1,\ldots,Y_n]$ be the polynomial ring over the field $\mathbb{K}$ in variables $Y_1 < \cdots < Y_n$. Let $p \in \mathbb{K}[Y]$ be a non-constant polynomial. The leading coefficient and the degree of p regarded as a univariate polynomial in Y_i will be denoted by $\mathrm{lc}(p, Y_i)$ and $\deg(p, Y_i)$ respectively. The greatest variable appearing in p is called the main variable denoted by $\mathrm{mvar}(p)$. The degree, the leading coefficient, and the leading monomial of p regarding as a univariate polynomial in $\mathrm{mvar}(p)$ are called the main degree, the initial, and the rank of p; they are denoted by $\mathrm{mdeg}(p)$, $\mathrm{init}(p)$ and $\mathrm{rank}(p)$ respectively.

Let $F\subset\mathbb{K}[Y]$ be a finite polynomial set. Denote by $\langle F\rangle$ the ideal it generates in $\mathbb{K}[Y]$ and by $\sqrt{\langle F\rangle}$ the radical of $\langle F\rangle$. Let h be a polynomial in $\mathbb{K}[Y]$, the saturated ideal $\langle F\rangle : h^{\infty}$ of $\langle F\rangle$ w.r.t h, is the set

$$\{q\in\mathbb{K}[Y]\mid \exists m\in\mathbb{N} \text{ s.t. } h^m q\in\langle F\rangle\},$$

which is an ideal in $\mathbb{K}[Y]$.

A polynomial $p\in\mathbb{K}[Y]$ is a zerodivisor modulo $\langle F\rangle$ if there exists a polynomial q such that pq is zero modulo $\langle F\rangle$, and q is not zero modulo $\langle F\rangle$. The polynomial is regular modulo $\langle F\rangle$ if it is neither zero, nor a zerodivisor modulo $\langle F\rangle$. Denote by $\mathbf{V}(F)$ the zero set (or solution set, or algebraic variety) of F in $\bar{\mathbb{K}}^n$, where $\bar{\mathbb{K}}$ is an algebraic closure of $\mathbb{K}$. For a subset $W\subset\bar{\mathbb{K}}^n$, denote by $\overline{W}$ its closure in the Zariski topology, that is the intersection of all algebraic varieties $\mathbf{V}(G)$ containing W for all $G\subset\mathbb{K}[Y]$.

Let $T\subset\mathbb{K}[Y]$ be a triangular set, that is a set of non-constant polynomials with pairwise distinct main variables. Denote by $\mathrm{mvar}(T)$ the set of main variables of $t\in T$. A variable in Y is called algebraic w.r.t. T if it belongs to $\mathrm{mvar}(T)$, otherwise it is called free w.r.t. T. For a variable $v\in Y$, we denote by $T_{<v}$ (resp. $T_{>v}$) the subsets of T consisting of the polynomials t with main variable less than (resp. greater than) v. If $v\in\mathrm{mvar}(T)$, we say T_v is defined. Moreover, we denote by T_v the polynomial in T whose main variable is v, by $T_{\leqslant v}$ the set of polynomials in T with main variables less than or equal to v and by $T_{\geqslant v}$ the set of polynomials in T with main variables greater than or equal to v.

Definition 1 Let p, $q\in\mathbb{K}[Y]$ be two nonconstant polynomials. We say $\mathrm{rank}(p)$ is smaller than $\mathrm{rank}(q)$ w.r.t Ritt ordering and we write, $\mathrm{rank}(p)<_r \mathrm{rank}(q)$ if one of the following assertions holds:

– $\mathrm{mvar}(p)<\mathrm{mvar}(q)$,

– $\mathrm{mvar}(p)=\mathrm{mvar}(q)$ and $\mathrm{mdeg}(p)<\mathrm{mdeg}(q)$.

Note that the partial order $<_r$ is a well ordering. Let $T\subset\mathbb{K}[Y]$ be a triangular set. Denote by rank(T) the set of $\mathrm{rank}(p)$ for all $p\in T$. Observe that any two ranks in rank (T) are comparable by $<_r$. Given another triangular set $S\subset\mathbb{K}[Y]$, with $\mathrm{rank}(S)\neq\mathrm{rank}(T)$, we write $\mathrm{rank}(T)<_r \mathrm{rank}(S)$ whenever the minimal element of the symmetric difference $(\mathrm{rank}(T)\setminus\mathrm{rank}(S))\cup(\mathrm{rank}(S)\setminus\mathrm{rank}(T))$ belongs to $\mathrm{rank}(T)$. By $\mathrm{rank}(T)\leqslant_r\mathrm{rank}(S)$, we mean either rank $(T)<\mathrm{rank}(S)$ or $\mathrm{rank}(T)=\mathrm{rank}(S)$. Note that any sequence of triangular sets, of which ranks strictly decrease w.r.t $<_r$, is finite.

Given a triangular set $T\subset\mathbb{K}[Y]$, denote by h_T the product of the initials of T (throughout the work we use this convention and when T consists of a single element g we write it as h_g for short). The quasi-component $\mathbf{W}(T)$ of T is $\mathbf{V}(T)/\mathbf{V}(h_T)$, in other words, the points of $\mathbf{V}(T)$ which do not cancel any of the initials of T. We denote by $\mathrm{Sat}(T)$ the saturated ideal of T: if T is empty then $\mathrm{Sat}(T)$ is defined as the trivial ideal $\langle 0\rangle$, otherwise it is the ideal $\langle T\rangle : h_T^{\infty}$.

Let $h\in\mathbb{K}[Y]$ be a polynomial and $F\subset\mathbb{K}[Y]$ a set of polynomials, we write

$$\boldsymbol{Z}(F,T,h) := (\boldsymbol{V}(F)\cap\boldsymbol{W}(T))\setminus\boldsymbol{V}(h).$$

When F consists of a single polynomial p, we use $\mathbf{Z}(p,T,h)$ instead of $\mathbf{Z}(\{p\},T,h)$; when F is empty we just write $\mathbf{Z}(T,h)$. By $\mathbf{Z}(F,T)$, we denote $\mathbf{V}(F)\cap\mathbf{W}(T)$.

Given a family of pairs $S=\{[T_i,h_i]|1\leqslant i\leqslant e\}$, where $T_i\subset\mathbb{K}[Y]$ is a triangular set and $h_i\in\mathbb{K}[Y]$ is a polynomial. We write

$$\mathbf{Z}(S) := \bigcup_{i=1}^{e}\mathbf{Z}(T_i,h_i).$$

We conclude this section with some well known properties of ideals and triangular sets. For a proper ideal $\mathcal{I}$, we denote by $\dim(\mathbf{V}(\mathcal{I}))$ the dimension of $\mathbf{V}(\mathcal{I})$.

Lemma 1 Let $\mathcal{I}$ be a proper ideal in $\mathbb{K}[Y]$ and $p\in\mathbb{K}[Y]$ be a polynomial regular w.r.t. $\mathcal{I}$. Then, either $\mathbf{V}(\mathcal{I})\cap\mathbf{V}(p)$ is empty or we have: $\dim(\mathbf{V}(\mathcal{I})\cap\mathbf{V}(p))\leqslant\dim(\mathbf{V}(\mathcal{I}))-1$.

Lemma 2 Let T be a triangular set in $\mathbb{K}[Y]$. Then, we have

$$\overline{\mathbf{W}(T)}\setminus\mathbf{V}(h_T)=\mathbf{W}(T) \text{ and}$$
$$\overline{\mathbf{W}(T)}\setminus\mathbf{W}(T)=\mathbf{V}(h_T)\cap\overline{\mathbf{W}(T)}.$$

Proof Since $\mathbf{W}(T)\subseteq\overline{\mathbf{W}(T)}$, we have

$$\mathbf{W}(T)=\mathbf{W}(T)\setminus\mathbf{V}(h_T)\subseteq\overline{\mathbf{W}(T)}\setminus\mathbf{V}(h_T).$$

On the other hand, $\overline{\mathbf{W}(T)}\subseteq\mathbf{V}(T)$ implies

$$\overline{\mathbf{W}(T)}\setminus\mathbf{V}(h_T)\subseteq\mathbf{V}(T)\setminus\mathbf{V}(h_T)=\mathbf{W}(T).$$

This proves the first claim. Observe that we have:

$$\overline{\mathbf{W}(T)}=\left(\overline{\mathbf{W}(T)}\setminus\mathbf{V}(h_T)\right)\cup\left(\overline{\mathbf{W}(T)}\cap\mathbf{V}(h_T)\right).$$

We deduce the second one.

Lemma 3 (Refs. [2,14]**)** Let T be a triangular set in $\mathbb{K}[Y]$. Then, we have

$$\mathbf{V}(\mathrm{Sat}(T))=\overline{\mathbf{W}(T)}.$$

Assume furthermore that $\mathbf{W}(T)\neq\varnothing$ holds. Then $\mathbf{V}(\mathrm{Sat}$ $(T))$ is a nonempty unmixed algebraic set with dimension

$n - |T|$. Moreover, if N is the free variables of T, then for every prime ideal $\mathcal{P}$ associated with Sat(T) we have

$$\mathcal{P} \cap [N] = \langle 0 \rangle.$$

3.1 Regular chain and regular system

Definition 2 (Regular chain) A triangular set $T \subset \mathbb{K}[Y]$ is a regular chain if one of the following conditions hold:

– either T is empty,

– or $T \backslash \{T_{\max}\}$ is a regular chain, where $T_{\max}$ is the polynomial in T with maximum rank, and the initial of $T_{\max}$ is regular w.r.t. Sat($T \backslash \{T_{\max}\}$).

It is useful to extend the notion of regular chain as follows.
Definition 3 (Regular system) A pair $[T,h]$ is a regular system if T is a regular chain, and $h \in \mathbb{K}[Y]$ is regular w. r.t Sat(T).

Remark 1 A regular system in a stronger sense was presented in Ref. [3]. For example, consider a polynomial system $[T,h]$ where $T = [Y_1 Y_4 - Y_2]$ and $h = Y_2 Y_3$. Then $[T,h]$ is still a regular system in our sense but not a regular system in Ref. [3]. We also note that in zero-dimension case (no free variables exist) the notion of a regular chain and that of a regular set in Ref. [3] are the same, Refs. [2,3] giving the details.

Proposition 1 For every regular system $[T,h]$ we have $\mathbf{Z}(T,h) \neq \emptyset$.
Proof Since T is a regular chain, by Lemma 3 we have $\mathbf{V}(\mathrm{Sat}(T)) \neq \emptyset$. By definition of a regular system, the polynomial hh_T is regular w.r.t Sat(T). Hence, by Lemma 1, the set $\mathbf{V}(hh_T) \cap \mathbf{V}(\mathrm{Sat}(T))$ either is empty, or has lower dimension than $\mathbf{V}(\mathrm{Sat}(T))$. Therefore, the set

$$\begin{aligned}&\mathbf{V}(\mathrm{Sat}(T))\backslash\mathbf{V}(hh_T)\\&=\mathbf{V}(\mathrm{Sat}(T))\backslash(\mathbf{V}(hh_T)\cap\mathbf{V}(\mathrm{Sat}(T)))\end{aligned}$$

is not empty. Finally, by Lemma 2, the set

$$\begin{aligned}\mathbf{Z}(T,h)&=\mathbf{W}(T)\backslash\mathbf{V}(h)\\&=\overline{\mathbf{W}(T)}\backslash\mathbf{V}(hh_T)\\&=\mathbf{V}(\mathrm{Sat}(T))\backslash\mathbf{V}(hh_T)\end{aligned}$$

is not empty.

3.2 Fundamental operations in TRIADE

Given a finite set of polynomial F, the TRIADE [15] algorithms can "decompose" the solution set $\mathbf{V}(F)$ of F by a set of regular chains $\{T_i | i = 1,\ldots,r\}$. Two kinds of decompositions are possible:

– Kalkbrener's triangular decomposition, that is:

$$\mathbf{V}(F) = \bigcup_{i=1}^{r} \overline{\mathbf{W}(T_i)}, \qquad (1)$$

– Lazard's triangular decomposition, that is:

$$\mathbf{V}(F) = \bigcup_{i=1}^{r} \mathbf{W}(T_i). \qquad (2)$$

We list below the specifications of the operations from TRIADE that we use in this paper. For simplicity, we use the notation $W \xrightarrow{D} (W_i, i = 1, \ldots, e)$ to denote the relation

$$W \subseteq \bigcup_{i=1}^{e} W_i \subseteq \overline{W}.$$

Below p, p_1, p_2 are polynomials, T and C are regular chains, D is a triangular set.

– **Regularize**(p,T) returns regular chains $T_1,\ldots,T_e$ such that

- $\mathbf{W}(T) \xrightarrow{D} (\mathbf{W}(T_i), i = 1,\ldots,e)$,
- for all $1 \leqslant i \leqslant e$, the polynomial p is either 0 or regular modulo Sat(T_i).

– **Intersect**(p,T) returns regular chains $T_1,\ldots,T_e$ such that we have

$$\begin{aligned}&\mathbf{V}(p)\cap\mathbf{W}(T)\subseteq\mathbf{W}(T_1)\cup\cdots\\&\cup\mathbf{W}(T_e)\subseteq\mathbf{V}(p)\cap\overline{\mathbf{W}(T)}.\end{aligned}$$

– **Extend**($C \cup D$) returns a set of regular chains $\{C_i | i = 1,\ldots,e\}$ such that

$$\mathbf{W}(C \cup D) \xrightarrow{D} (\mathbf{W}(C_i), i = 1, \ldots, e).$$

– Assume that p_1 and p_2 are two non-constant polynomials with the same main variable v, which is larger than any variable appearing in T, and assume that the initials of p_1 and p_2 are both regular w.r.t. Sat(T). Then, **GCD**(p_1,p_2,T) returns a sequence $([g_1,C_1],\ldots,[g_d,C_d],[\emptyset,D_1],\ldots,[\emptyset,D_e])$, where g_i are polynomials and C_i,D_i are regular chains such that

- $\mathbf{W}(T) \xrightarrow{D} (\mathbf{W}(C_1),\ldots,\mathbf{W}(C_d),\mathbf{W}(D_1),\ldots,\mathbf{W}(D_e))$,
- dim $\mathbf{V}(\mathrm{Sat}(C_i))$ = dim $\mathbf{V}(\mathrm{Sat}(T))$ and dim $\mathbf{V}(\mathrm{Sat}(D_j))$ < dim $\mathbf{V}(\mathrm{Sat}(T))$, for $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant e$,
- the leading coefficient of g_i w.r.t. v is regular w.r.t. Sat(C_i),
- $g_i = u_i p_1 + v_i p_2 \bmod \mathrm{Sat}(C_i)$ for some polynomials u_i and v_i,
- if g_i is not constant and its main variable is v, then p_1 and p_2 belong to Sat($C_i \cup \{g\}$).

Based on the operations **Regularize** and **Intersect**, we deduce a general intersection operation which decomposes a regular chain to regular systems, according to an equality and an inequality.

Lemma 4 Let p and h be polynomials and T be a regular chain. There exists an operation **IntersectGeneral**(p,T,h) returning a set of regular chains $\{T_1,\ldots,T_e\}$ such that:

(i) h is regular w.r.t $\mathrm{Sat}(T_i)$ for all i;

(ii) $\mathbf{Z}(p,T,h) \subseteq \bigcup_{i=1}^{e} \mathbf{Z}(T_i,h) \subseteq (\mathbf{V}(p) \cap \overline{\mathbf{W}(T)}) \setminus \mathbf{V}(h)$;

(iii) Moreover, if the product of initials h_T of T divides h, then

$$\mathbf{Z}(p,T,h) = \bigcup_{i=1}^{e} \mathbf{Z}(T_i,h).$$

Proof Intersect(p,T) returns regular chains $T_1,\ldots,T_e$ such that

$$\mathbf{V}(p) \cap \mathbf{W}(T) \subseteq \mathbf{W}(T_1) \cup \cdots \cup \mathbf{W}(T_e) \subseteq \mathbf{V}(p) \cap \overline{\mathbf{W}(T)}. \tag{3}$$

Using **Regularize**(h,T_i) we can assume that (i) holds. (3) clearly implies (ii). The conclusion (iii) follows from Lemma 2.

4 Representations of constructible sets

The constructible set [16,17] is a classical concept in elimination theory. In this section, we present two types of representations for constructible sets in $\bar{\mathbb{K}}^n$.

Definition 4 (Constructible set) A constructible subset of $\bar{\mathbb{K}}^n$ is any 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 over $\mathbb{K}$. Let $\mathcal{F}$ be the set of all constructible subsets of $\bar{\mathbb{K}}^n$ w.r.t $\mathbb{K}$. From Exercise 3.18 in Ref. [17], we have

– all open algebraic sets are in $\mathcal{F}$;

– the complement of an element in $\mathcal{F}$ is in $\mathcal{F}$;

– the intersection of two elements in $\mathcal{F}$ is in $\mathcal{F}$.

Moreover, these three properties describe exactly all constructible sets.

Given a set of polynomial F and $f \in \mathbb{K}[Y]$, we denote $\mathbf{D}(F,f)$ the difference of $\mathbf{V}(F)$ and $\mathbf{V}(f)$, which is also called a basic constructible set. If F is the empty set, then we write $\mathbf{D}(f)$ for short. Note that for a regular system in Ref. [3], $\mathbf{D}(T,h) = \mathbf{Z}(T,h)$ holds.

4.1 Gröbner basis representation

Now Gröbner bases have become a standard tool to deal with algebraic sets; and they can be applied to manipulate constructible sets as well. Given a constructible set C, according to the definition, one can represent C by a unique sequence of closed algebraic sets whose defining ideals naturally can be characterized by their reduced Gröbner bases [18].

However, the constructible set is a geometrical object intrinsically. We pay extra cost to manipulate them, since it is very hard to compute the intersection of two ideals and even to compute the radical ideal of an ideal. Whatsoever, there exist effective algorithms to manipulate constructible sets. We shall use regular systems to do the same jobs in a more efficient manner.

4.2 Regular system representation

In this section, we show that (Theorem 2) every constructible set C can be represented by a finite set of regular systems $\{[T_i,h_i] | i = 1,\ldots,e\}$, that is:

$$C = \bigcup_{i=1}^{e} \mathbf{Z}(T_i,h_i).$$

Combining with Lemma 1, we know that if a regular system representation of a constructible set is empty then C is an empty set. This fact leads to an important application of verifying polynomial system solver. The proof of Theorem 2 is partially constructible and relies on an algorithm called **Difference**, presented in Section 5. As an immediate consequence of the specifications of this algorithm, we obtain the following theorems.

Theorem 1 Given two regular systems $[T,h]$ and $[T',h']$, there is an algorithm to compute the regular system representations of:

(1) the difference $\mathbf{Z}(T,h) \setminus \mathbf{Z}(T',h')$;

(2) the intersection $\mathbf{Z}(T,h) \cap \mathbf{Z}(T',h')$.

Proof (1) follows from the algorithm **Difference**. Note that given two sets A and B, $A \cap B = A \setminus (A \setminus B)$. (2) follows from a successive call to **Difference**.

Theorem 2 Every constructible set can be represented by a finite set of regular systems.

Proof Consider the following family $\tilde{\mathcal{F}}$ of subsets of $\bar{\mathbb{K}}^n$:

$$\tilde{\mathcal{F}} = \left\{ S | S = \bigcup_{i=1}^{e} \mathbf{Z}(T_i,p_i) \right\},$$

where $[T_i,p_i]$ are regular systems. First, every open subset can be decomposed into the finite union of open subsets $\mathbf{D}(f)$ which can be represented by the empty regular chain and f. Hence $\tilde{\mathcal{F}}$ contains all open subsets. Second, consider two elements S and T in $\tilde{\mathcal{F}}$; and assume that

$$S = \bigcup_{i=1}^{e} \mathbf{Z}(S_i,p_i) \quad \text{and} \quad T = \bigcup_{j=1}^{f} \mathbf{Z}(T_j,q_j).$$

We have

$$S \bigcap T = \bigcup_{i=1}^{e} \bigcup_{j=1}^{f} \left(\mathbf{Z}(S_i,p_i) \bigcap \mathbf{Z}(T_j,q_j) \right).$$

By Theorem 1, $S \cap T$ has a regular system representation, that is to say, $S \cap T \in \tilde{\mathcal{F}}$. By induction, any finite intersection of elements of $\tilde{\mathcal{F}}$ is in $\tilde{\mathcal{F}}$. Finally, we shall prove that the complement of an element in $\tilde{\mathcal{F}}$ is in $\tilde{\mathcal{F}}$. Essentially, we only need to show that for each $1 \leqslant i \leqslant e$, $\mathbf{Z}(S_i,p_i)^c$ is in $\tilde{\mathcal{F}}$. Indeed,

$$\begin{aligned}\mathbf{Z}(S_i,p_i)^c &= \mathbf{W}(S_i)^c \bigcup \mathbf{V}(p_i)\\ &= \mathbf{V}(S_i)^c \bigcup \mathbf{V}(p_i h_{S_i})\end{aligned}$$

is in $\tilde{\mathcal{F}}$, since both $\mathbf{V}(S_i)^c$ and $\mathbf{V}(p_i h_{S_i})$ have regular system representations.

5 The Difference algorithm

In this section, we present an algorithm to compute the set theoretical difference of two constructible sets given by regular systems. Here we contribute a sophisticated algorithm, which heavily exploits the structure and properties of regular chains.

Two procedures, **Difference** and **DifferenceLR**, are involved in order to achieve this goal. Their specification and pseudo-code can be found below. The correctness and termination of these algorithms are established in Ref. [19]. For the pseudo-code, we use the MAPLE syntax. However, each of the two functions below returns a sequence of values. Individual value or sub-sequences of the returned sequence are thrown to the flow of **output** by means of an **output** statement. Hence an output statement does not cause the termination of the function execution.

Algorithm① Difference $([T,h],[T',h'])$

Input $[T,h],[T',h']$ two regular systems.

Output Regular systems $\{[T_i,h_i] | i = 1,\ldots,e\}$ such that:

$$\mathbf{Z}(T,h) \backslash \mathbf{Z}(T',h') = \bigcup_{i=1}^{e} \mathbf{Z}(T_i,h_i).$$

Algorithm② DifferenceLR (L,R)

Input $L = \{[L_i,f_i] | i = 1,\ldots,r\}$ and $R = \{[R_j,g_j] | j = 1,\ldots,s\}$ two lists of regular systems.

Output Regular systems $\{[T_i,h_i] | i = 1,\ldots,e\}$ such that:

$$\left(\bigcup_{i=1}^{r} \mathbf{Z}(L_i,f_i))\right) \backslash \left(\bigcup_{j=1}^{s} \mathbf{Z}(R_j,g_j)\right) = \bigcup_{i=1}^{e} \mathbf{Z}(T_i,h_i).$$

Theorem 3 The algorithms **Difference** and **DifferenceLR** terminate and satisfy the specifications as stated.

The proof of Theorem 3 is quite involved, and a detailed proof is in Ref. [19]. Alternatively, we give a naive method to realize the **Difference** algorithm here. For two regular systems $[T_1,h_1]$ and $[T_2,h_2]$, the following formula,

$$\begin{aligned}\mathbf{Z}(T_1,h_1) \backslash \mathbf{Z}(T_2,h_2) &= \left(\mathbf{Z}(T_1,h_1) \bigcap \mathbf{V}(T_2)^c\right) \cup \left(\mathbf{Z}(T_1,h_1) \bigcap \mathbf{V}(h_2 h_{T_2})\right)\\ &= \underbrace{\left(\bigcup_{f \in T_2} \mathbf{Z}(T_1,h_1) \backslash \mathbf{V}(f)\right)}_{\text{Task A}} \bigcup \underbrace{(\mathbf{Z}(T_1,h_1) \cap \mathbf{V}(h_2 h_{T_2}))}_{\text{Task B}},\end{aligned} \tag{4}$$

provides a method to compute the difference of the zero sets of two regular systems, where Task A is achieved by a call to **IntersectGeneral**$(0,T_1,fh_1h_{T_1})$ and Task B is achieved by a call to **IntersectGeneral**$(h_2h_{T_2},T_1,h_1h_{T_1})$. However, this method completely ignores the structure of $[T_2,h_2]$ (a regular system). Algorithm 1 is more subtle which exploits heavily the structure of $[T_2,h_2]$. In broad words, the procedure processes as follows.

(1) If $\mathrm{Sat}(T_1) = \mathrm{Sat}(T_2)$ holds, computations reduce to elementary manipulations of zero sets.

(2) Otherwise, let v be the largest variable such that $\mathrm{Sat}(T_{1,<v}) = \mathrm{Sat}(T_{2,<v})$. Let G be a **GCD** of $T_{1,v}$ and $T_{2,v}$ modulo $\mathrm{Sat}(T_{1,<v})$. If G is not constant and has main variable v, computations split into cases where either one can conclude easily or a recursive call to the procedure can be made. If G is constant or has main variable less than v, one can also easily conclude.

In Section 7, we will see that ignoring the structure of $[T_2,h_2]$ will result in bad performance. For some systems, we can solve them but we cannot verify via the naive difference algorithm.

Algorithm 1 Difference($[T,h]$, $[T',h']$)

```
1:  if $\mathrm{Sat}(T) = \mathrm{Sat}(T')$ then
2:      output IntersectGeneral($h'h_{T'}, T, hh_T$)
3:  else
4:      Let $v$ be the largest variable s.t. $\mathrm{Sat}(T_{<v}) = \mathrm{Sat}(T'_{<v})$
5:      if $v \in \mathrm{mvar}(T')$ and $v \notin \mathrm{mvar}(T)$ then
6:          $p' \leftarrow T'_v$
7:          output $[T, hp']$
8:          output DifferenceLR(IntersectGeneral($p', T, hh_T$), $[T',h']$)
9:      else if $v \notin \mathrm{mvar}(T')$ and $v \in \mathrm{mvar}(T)$ then
10:         $p \leftarrow T_v$
11:         output DifferenceLR($[T,h]$, IntersectGeneral($p, T', h'h_{T'}$))
12:     else
13:         $p \leftarrow T_v$
14:         $\mathcal{G} \leftarrow GCD(T_v, T'_v, T_{<v})$
15:         if $|\mathcal{G}| = 1$ then
16:             Let $(g, C) \in \mathcal{G}$
17:             if $g \in \mathbb{K}$ then
```

```
18:             output $[T,h]$
19:         else if mvar$(g) < v$ then
20:             output $[T,gh]$
21:             output DifferenceLR(IntersectGeneral
                    $(g,T,hh_T)$, $[T',h']$)
22:         else if mvar$(g) = v$ then
23:             if mdeg$(g)$ = mdeg$(p)$ then
24:                 $D'_p \leftarrow T'_{<v} \cup \{p\} \cup T'_{>v}$
25:                 output Difference($[T,h]$, $\left[D'_p, h'h_{T'}\right]$)
26:             else if mdeg$(g) <$ mdeg$(p)$ then
27:                 $q \leftarrow$ pquo$(p,g,C)$
28:                 $D_g \leftarrow C \cup \{g\} \cup T_{>v}$
29:                 $D_q \leftarrow C \cup \{q\} \cup T_{>v}$
30:                 output Difference($[D_g,hh_T]$, $[T',h']$)
31:                 output Difference($[D_q,hh_T]$, $[T',h']$)
32:                 output DifferenceLR(IntersectGeneral
                        $(h_g,T,hh_T)$, $[T',h']$)
33:             end if
34:         end if
35:     else if $|\mathcal{G}| \geqslant 2$ then
36:         for $(g,C) \in \mathcal{G}$ do
37:             if $|C| > |T_{<v}|$ then
38:                 for $E \in$ Extend$(C,T_{\geqslant v})$ do
39:                     for $D \in$ Regularize$(hh_T,E)$ do
40:                         if $hh_T \notin$ Sat$(D)$ then
41:                             output Difference($[D,hh_T]$, $[T',$
                                    $h']$)
42:                         end if
43:                     end for
44:                 end for
45:             else
46:                 output Difference($[C \cup T_{\geqslant v},hh_T]$, $[T',$
                        $h']$)
47:             end if
48:         end for
49:     end if
50:   end if
51: end if
```

Algorithm 2 DifferenceLR(L,R)

```
1:  if $L = \varnothing$ then
2:      output $\varnothing$
3:  else if $R = \varnothing$ then
4:      output $L$
5:  else if $|R| = 1$ then
6:      Let $[T',h'] \in R$
7:      for $[T,h] \in L$ do
8:          output Difference($[T,h]$, $[T',h']$)
9:      end for
10: else
11:     while $R \neq \varnothing$ do
12:         Let $[T',h'] \in R$, $R \leftarrow R \setminus \{[T',h']\}$
13:         $S \leftarrow \varnothing$
14:         for $[T,h] \in$ L do
15:             $S \leftarrow S \cup$ Difference($[T,h]$, $[T',h']$)
16:         end for
17:         $L \leftarrow S$
18:     end while
19:     output $L$
20: end if
```

6 Verification of triangular decompositions

In this section, we describe how to verify the output from a triangular decomposition solver. For verification of triangular decomposition in Kalkbrener's sense, it is still unknown whether we can circumvent Gröbner basis computations. However, in Lazard's sense, we present two methods, based on Gröbner bases and regular systems, respectively.

6.1 Verification with Gröbner bases

The following two lemmas state the Gröbner basis methods to verify whether two basic constructible sets are equal or not.

Lemma 5 Let $\{F,f\}$ and $\{G_0,g_0\}$ be two polynomial systems. The following statements are equivalent

1) $\mathbf{D}(F,f) \setminus \mathbf{D}(G_0,g_0) \subseteq \bigcup_{i=1}^{r} \mathbf{D}(G_i,g_i)$.

2) For every $\{i_1,\ldots,i_s\} \subseteq \{0,\ldots,r\}$, $0 \leqslant s \leqslant r$,

$$\sqrt{\langle F \cap \{g_{i_1},\ldots,g_{i_s}\}\rangle} \supseteq \prod_{k \in \{0,\ldots,r\} \setminus \{i_1,\ldots,i_s\}} \langle f \rangle \langle G_k \rangle. \quad (5)$$

Proof 1) is equivalent to $\mathbf{D}(F,f) \subseteq \bigcup_{i=0}^{r} \mathbf{D}(G_i,g_i)$.

$$\mathbf{D}(F,f) \bigcap \left(\bigcap_{i=0}^{r} \mathbf{D}(G_i,g_i)^c \right) = \varnothing.$$

Using the distributive property, we deduce that 1) is equivalent to

$$(\mathbf{D}(F,f) \cap \mathbf{V}(g_{i_1},\ldots,g_{i_s})) \cap \left(\bigcap_{k \in \{0,\ldots,r\} \setminus \{i_1,\ldots,i_s\}} \mathbf{V}(G_k)^c \right) = \varnothing,$$

for all subsets $\{i_1,\ldots,i_s\}$ of $\{0,\ldots,r\}$. The proof easily follows.

Lemma 6 Let $\{F,f\}$ and $\{G,g\}$ be two polynomial systems. The following statements are equivalent

1) $\mathbf{D}(F,f) \setminus \mathbf{D}(G,g) \supseteq \bigcup_{i=1}^{r} \mathbf{D}(H_i,h_i)$.

2) For all $1 \leqslant i \leqslant r$, we have

$$\begin{gathered} h_i g \in \sqrt{\langle H_i \cup G \rangle}, h_i \in \sqrt{\langle H_i, f \rangle}, \text{ and} \\ \langle h_i \rangle \langle F \rangle \subset \sqrt{\langle H_i \rangle}. \end{gathered} \quad (6)$$

Proof 1) holds if and only if for each $1 \leqslant i \leqslant$ r we have

$$\begin{cases} \mathbf{D}(H_i,h_i) \bigcap \mathbf{D}(F,f)^c = \varnothing, \\ \mathbf{D}(H_i,h_i) \bigcap \mathbf{D}(G,g) = \varnothing, \end{cases}$$

which holds if and only if

$$\begin{cases} \mathbf{V}(H_i)\bigcap\mathbf{V}(h_i)^c\bigcap\mathbf{V}(F)^c=\varnothing, \\ \mathbf{V}(H_i)\bigcap\mathbf{V}(h_i)^c\bigcap\mathbf{V}(f)=\varnothing, \\ \mathbf{V}(H_i)\bigcap\mathbf{V}(h_i)^c\bigcap\mathbf{V}(G)\bigcap\mathbf{V}(g)^c=\varnothing. \end{cases}$$

The proof easily follows.

The above general lemmas can be used to check if the output from the algorithm Difference is correct or not. In particular, they can be applied to check if a triangular decomposition is valid or not by comparing the input system and one triangular decomposition. We naively implement them using MAPLE package PolynomialIdeals.

6.2 Verification with the Difference algorithm

Given two Lazard's triangular decompositions $\{T_i|i=1,\ldots,e\}$ and $\{S_j|j=1,\ldots,f\}$ of a polynomial system. Checking $\bigcup_{i=1}^{e}\mathbf{W}(T_i)=\bigcup_{j=1}^{f}\mathbf{W}(S_j)$ amounts to checking both

$$\left(\bigcup_{i=1}^{e}\mathbf{W}(T_i)\right)\Bigg\backslash\left(\bigcup_{j=1}^{f}\mathbf{W}(S_j)\right) \text{ and}$$

$$\left(\bigcup_{j=1}^{f}\mathbf{W}(S_j)\right)\Bigg\backslash\left(\bigcup_{i=1}^{e}\mathbf{W}(T_i)\right)$$

being empty. In turn, after computing the regular system representations of above two constructible sets, according to Theorem 3, we solve the verification problem with the algorithm DifferenceLR in Lazard's sense.

7 Experimentation

We have implemented a verifier, named Diff-verifier, according to the DifferenceLR algorithm proposed in Section 5, and it has been implemented in MAPLE 11 based on the REGULARCHAINS library. To verify the effectiveness of our Diff-verifier, we have also implemented

Table 1 Features of the polynomial systems

					number of components			
Sys	name	n	d	dimension	MAPLE Triangularize	ALDOR TRIADE server	Epsilon RegSer	Epsilon SimSer
1	Montes S1	4	2	[2,2,1]	3	3	3	3
2	Montes S2	4	3	[0]	1	1	1	1
3	Montes S3	3	3	[1,1]	2	2	2	3
4	Montes S4	4	2	[0]	1	1	1	1
5	Montes S6	4	3	[2,2,2]	3	3	3	3
6	Montes S7	4	3	[1]	2	2	3	6
7	Montes S8	4	12	[2,1]	2	2	6	6
8	Alonso	7	4	[3]	3	3	3	4
9	Raksanyi	8	3	[4]	4	4	4	10
10	YangBaxter Rosso	6	3	[4,3,3,1,1,1,1]	7	7	4	13
11	l-3	4	3	[0,0,0,0,0,0,0,0,0,0,0,0,0]	25	13	8	8
12	Caprasse	4	4	[0,0,0,0,0]	15	5	4	4
13	Reif	16	3	[]	0	0	0	0
14	Buchberger WuWang	5	3	[2]	3	3	3	4
15	DonatiTraverso	4	31	[1]	6	3	3	3
16	Wu-Wang.2	13	3	[1,1,1,1,1]	5	5	5	5
17	Hairer-2-BGK	13	4	[2]	4	4	5	6
18	Montes S5	8	3	[4]	4	4	4	10
19	Bronstein	4	3	[1]	4	2	4	9
20	Butcher	8	4	[3,3,3,2,2,0]	7	6	6	6
21	genLinSyst-2-2	8	2	[6]	11	11	11	11
22	genLinSyst-3-2	11	2	[8]	17	18	18	18
23	Gerdt	7	4	[3,2,2,2,1,1]	7	6	10	10
24	Wang93	5	3	[1]	5	4	6	7
25	Vermeer	5	5	[1]	5	4	12	14
26	Gonnet	5	2	[3,3,3]	3	3	9	9
27	Neural	4	3	[1,1]	4	3	–	–
28	Noonburg	4	3	[1,1]	4	3	–	–
29	KdV	26	3	[12,12,11,11,11,11,11]	7	7	–	–
30	Montes S12	8	2	[4]	22	17	23	–
31	Pappus	12	2	[6,6,6,6,6,6,6,6,6]	124	129	156	–

Table 2 Solving timings in second of the four methods

Sys	Maple Triangularize	ALDOR TRIADE server	Epsilon RegSer	Epsilon SimSer
1	0.104	0.164	0.01	0.03
2	0.039	0.204	0.03	0.02
3	0.069	0.06	0.019	0.111
4	0.510	0.072	0.049	0.03
5	0.052	0.096	0.03	0.03
6	0.150	0.06	0.09	5.14
7	0.376	0.072	0.2	1.229
8	0.204	0.065	0.109	0.16
9	0.460	0.066	0.141	0.481
10	1.252	0.108	0.069	0.21
11	5.965	0.587	1.53	2.91
12	2.426	0.167	1.209	2.32
13	123.823	1.886	1.979	2.36
14	0.2	0.101	0.049	0.109
15	2.641	0.08	0.439	0.7
16	105.835	1.429	5.49	6.14
17	23.453	0.688	1.76	1.679
18	0.484	0.078	0.13	0.471
19	0.482	0.071	0.24	1.000
20	9.325	0.442	1.689	2.091
21	0.557	0.096	0.13	0.21
22	1.985	0.173	0.431	0.411
23	4.733	0.499	3.5	4.1
24	7.814	5.353	2.18	30.24
25	26.533	0.580	4.339	60.65
26	3.983	0.354	2.18	2.48
27	15.879	1.567	–	–
28	15.696	1.642	–	–
29	9245.442	49.573	–	–
30	17.001	0.526	2.829	–
31	79.663	4.429	11.78	–

another verifier, named GB-verifier, applying Lemma 5 and Lemma 6, on top of the PolynomialIdeals package in MAPLE 11.

We use these two verifiers to examine four polynomial system solvers herein. They are the Triangularize function in the RegularChains library [8], the TRIADE server in ALDOR on top of the BasicMath library [9], the RegSer function and the SimSer function in Epsilon [10] implemented in MAPLE. The first two solvers solve a polynomial system into regular chains by means of the TRIADE algorithm [15]. They can work in both Lazard's sense and Kalkbrener's sense. In this work, we use the options for solving in Lazard's sense. The RegSer function decomposes a polynomial system into regular systems in a strong sense, and the SimSer function decomposes a polynomial system into simple systems. They adopt the elimination methods in Ref. [13].

The problems used in this benchmark are chosen from Refs. [11–13]. In Table 1, for each system, we give the dimension sequence of the triangular decomposition computed in Kalkbrener's sense by the TRIADE algorithm. The number of variables is denoted by n, and d is the maximum degree of a monomial. We also give the number of components in the solution set for each of the methods we are studying.

Table 2 gives the timing of each problem solved by the four methods. In this study, due to the current availability of Epsilon, the timings obtained by the RegSer and the SimSer are performed in Maple 8 on Intel Pentium 4 machines (1.60 GHz CPU, 513 MB memory and Red Hat Linux 3.2.2-5). All the other timings are run on Intel Pentium 4 (3.20 GHz CPU, 2.0 GB total memory, and Linux 4.0.0-9), and the MAPLE version used is 11. The TRIADE server is a stand-alone executable program compiled from a program in ALDOR.

Table 3 summarizes the timings of GB-verifier for verifying the solutions of the four methods. Table 3 gives also the timings of Diff-verifier for checking the solutions by MAPLE Triangularize vs. ALDOR TRIADE server, MAPLE Triangularize versus Epsilon RegSer, and Epsilon RegSer versus Epsilon SimSer. For the case where there is a time, the verifying result is also true. The '–' denotes the case where the test stalls by either reaching the time limit of 43200 seconds or causing a memory failure.

Based on Eq. (4) in Section 5, we implement the Difference operation naively, and we call it Naive-diff-verifier. From the Table 4 we can see clearly that, for most problems, the Diff-verifier performs much better than Naive-diff-verifier, especially for hard problems.

Table 3 Timings of GB-verifier and Diff-verifier

	GB-verifier timing/s				Diff-verifier timing/s		
sys	Maple Triangularize (M.T.)	ALDOR TRIADE server (A.T.)	Epsilon RegSer (E.R.)	Epsilon SimSer (E.S.)	M.T. vs A.T.	M.T. vs E.R.	E.R. vs E.S.
1	0.556	0.526	0.518	0.543	0.188	0.238	0.217
2	0.128	0.127	0.129	0.131	0.012	0.010	0.010
3	0.584	0.575	0.585	2.874	0.067	0.088	0.326
4	0.104	0.133	0.139	0.137	0.018	0.017	0.018
5	1.484	1.472	1.457	1.469	0.198	0.178	0.190
6	76.596	72.374	71.853	–	2.010	2.390	12.591
7	0.616	0.601	4.501	4.536	0.191	0.404	0.492
8	–	–	–	–	0.571	0.677	0.925
9	–	–	–	–	4.257	4.454	7.884
10	–	–	–	–	6.555	8.824	9.037
11	–	–	–	–	5.341	3.564	1.997
12	–	58.332	33.469	35.213	1.506	1.657	2.354
13	–	–	–	–	0.000	0.000	0.000
14	1.96	1.937	2.165	5.739	0.617	0.661	0.722
15	330.317	–	–	–	1.689	3.095	2.870
16	10466.587	–	–	–	1.340	0.795	0.773
17	–	–	–	–	1.883	2.272	4.903
18	–	–	–	–	4.450	4.596	8.063
19	1.544	0.717	5.046	–	2.162	6.382	41.374
20	–	–	–	–	5.683	5.113	5.949
21	–	–	–	–	6.595	6.621	4.441
22	–	–	–	–	21.689	17.943	11.503
23	–	–	–	–	4.073	5.071	5.775
24	–	–	–	–	1064.127	636.221	707.668
25	–	–	–	–	817.499	1519.858	1585.095
26	–	–	–	–	0.554	1.276	1.741
27	11383.335	–	–	–	1072.199	–	–
28	–	–	–	–	1248.353	–	–
29	–	–	–	–	5.418	–	–
30	–	–	–	–	428.503	706.854	–
31	–	–	–	–	8071.055	9800.086	–

This experimentation results illustrate that verifying a polynomial solver is a truly difficult task. The GB-verifier is very costly in terms of CPU time and memory. It only succeeds for some easy examples. Assuming that the GB-verifier is reliable, for the examples it succeeds, the Diff-verifier agrees with its results by pairwise checking, while it takes much less time. This shows the efficiency of our Diff-verifier. Moreover, the Diff-verifier succeeds in computing the difference for any pair of output of the four solvers. Therefore, our new approach can verify the solution set of all test

Table 4 Timings of Naive-diff-verifier and Diff-verifier for M.T. vs A.T.

Sys	Naive-diff-verifier timing/s	Diff-verifier timing/s	Sys	Naive-diff-verifier timing/s	Diff-verifier timing/s
1	0.027	0.188	17	10876.470	1.883
2	0.003	0.012	18	5.498	4.450
3	0.075	0.067	19	7.491	2.162
4	0.010	0.018	20	450.342	5.683
5	0.049	0.198	21	158.879	6.595
6	2.146	2.010	22	4450.023	21.689
7	0.111	0.191	23	11.415	4.073
8	1.815	0.571	24	25047.768	1064.127
9	5.342	4.257	25	–	817.499
10	58.938	6.555	26	0.373	0.554
11	–	5.341	27	2466.459	1072.199
12	–	1.506	28	2464.389	1248.353
13	0.000	0.000	29	316.925	5.418
14	3.254	0.617	30	–	428.503
15	11.813	1.689	31	–	8071.055
16	11.374	1.340			

polynomial systems that at least two of our four solvers can solve, which serves well for our purpose.

Furthermore, the tests also show that the Diff-verifier can verify quite difficult problems. Therefore, the tests indicate that all four solvers are solving tools with a high probability of correctness, since the checking results would not agree to each other otherwise.

References

1. Grabmeier J, Kaltofen E, Weispfenning V. Computer Algebra Handbook. Berlin: Springer, 2003
2. Aubry P, Lazard D, Moreno Maza M. On the theories of triangular sets. Journal of Symbolic Computation, 1999, 28(1–2): 105–124
3. Wang D. Computing triangular systems and regular systems. Journal of Symbolic Computation, 2000, 30(2): 221–236
4. Donati L, Traverso C. Experimenting the Gröbner basis algorithm with the ALPI system. In: Proceedings of ISSAC. New York: ACM Press, 1989, 192–198
5. Aubry P, Moreno Maza M. Triangular sets for solving polynomial systems: a comparative implementation of four methods. Journal of Symbolic Computation, 1999, 28(1–2): 125–154
6. Backelin J, Fröberg R. How we proved that there are exactly 924 cyclic 7-roots. In: Watt S M, Proceedings of ISSAC. New York: ACM Press, 1991, 103–111
7. Sit W. Computations on quasi-algebraic sets. In: Proceedings of IMACS ACA, 1998
8. Lemaire F, Moreno Maza M, Xie Y. The regularChains library. In: Kotsireas I S, ed. Proceedings of Maple Conference 2005. 2005, 355–368
9. The Computational Mathematics Group. The BasicMath library NAG Ltd, Oxford, UK, 1998. http://www.nag.co.uk/projects/FRISCO.html
10. Wang D. Epsilon 0.618. http://www-calfor.lip6.fr/wang/epsilon
11. Manubens M, Montes A. Improving DISPGB Algorithm Using the Discriminant Ideal, 2006. http://www.citebase.org/abstract?id=oai:arXiv.org:math/0601763
12. The SymbolicData Project, 2000–2006. http://www.SymbolicData.org
13. Wang D. Elimination Methods. Berlin: Springer, 2001
14. Boulier F, Lemaire F, Moreno Maza M. Well known theorems on triangular systems. In: Proceedings of Transgressive Computing 2006. Spain: University of Granada, 2006
15. Moreno Maza M. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/~moreno
16. Eisenbud D. Commutative Algebra. GTM 150. Berlin: Springer, 1994
17. Hartshorne R. Algebraic Geometry. Berlin: Springer-Verlag, 1997
18. O'Halloran J, Schilmoeller M. Gröbner bases for constructible sets. Journal of Communications in Algebra, 2002, 30(11): 5479–5483
19. Chen C, Golubitsky O, Lemaire F, et al. Comprehensive triangular decomposition. In: Proceedings of Computer Algebra in Scientific Computing. Berlin: Springer, LNCS, 2007, 4770: 73–101