J. Symbolic Computation (1999) **28**, 125–154
Article No. jsco.1998.0270
Available online at http://www.idealibrary.com on IDEAL®

Triangular Sets for Solving Polynomial Systems: a Comparative Implementation of Four Methods

PHILIPPE AUBRY[†§] AND MARC MORENO MAZA[†‡¶]

[†] *LIP6, Université Paris 6, 4 Place Jussieu, 75252 Paris Cedex 05, France*
[‡] *Computational Mathematics Group, NAG Ltd, Jordan Hill Rd, Oxford OX2 8DR, U.K.*

Four methods for solving polynomial systems by means of triangular sets are presented and implemented in a unified way. These methods are those of Wu (1987), Lazard (1991), Kalkbrener (1991) and Wang (1993b). They are compared on various examples with the emphasis on efficiency, conciseness and legibility of the output.

© 1999 Academic Press

1. Introduction

In this paper, we are concerned with the following problem: given a finite family F of multivariate polynomials over a field $\mathbf{k}$ with ordered variables $x_1 < x_2 < \cdots < x_n$, we want to describe the affine variety $\mathbf{V}(F)$ (i.e. the common zeroes of F over an algebraic closure of $\mathbf{k}$). Such a description is usually provided by a finite family $\{T_1, \ldots, T_r\}$ of polynomial sets with particular properties, a relation between the T_i and F, and an algorithm to compute the T_i from F. A well-developed method since Buchberger (1965) is the following: given an ordering on the monomials, choose for T_1 a Gröbner basis of the ideal generated by F and compute it by the Buchberger's algorithm.

Following the work of Ritt (1932, 1966), Wu (1986) introduced another way of solving algebraic systems which is the one we are concerned with in this paper. In this case each T_i is a polynomial set such that two distinct polynomials in T_i have distinct greatest variables. Such a T_i is called a triangular set. A point $\zeta \in \mathbf{V}(T_i)$ is called regular if for every $p \in T_i$ the point ζ does not cancel the initial of p (that is the leading coefficient of p regarded as a univariate polynomial in its greatest variable). Then, in Wu's method, the variety $\mathbf{V}(F)$ is the union of the regular zeroes of the T_i and this decomposition can be computed by Wu's CHRST-REM algorithm (Wu, 1987). This method has been investigated in many papers. Among them: (Chou, 1988; Chou and Gao, 1990, 1992; Gallo and Mishra, 1990; Wang, 1992, 1995). Wu's method is efficient for geometric problems where the degenerate solutions are not interesting. For general problems it seems to be difficult to obtain an efficient implementation and this method may produce superfluous triangular sets. Wu's algorithm, like Buchberger's, depends on many choices; moreover, its result is not uniquely defined.

[§]E-mail: `aubry@calfor.lip6.fr`
[¶]E-mail: `marc@nag.co.uk`

Lazard (1991) proposed a new method to obtain Wu-like decompositions for affine varieties. However, in this case, the definition of triangular sets has been strengthened (Definition 2.3) in order to guarantee non-redundant decompositions. Some details and proofs were not given in detail, especially on the subject of gcd computation, which is the main tool of the method. In Moreno Maza (1997), these questions are treated and a first implementation of Lazard's method is described and shown to be efficient.

Kalkbrener (1991) introduced another type of triangular sets called regular chains (Definition 2.2) together with another relation between F and the T_i. In this case $\mathbf{V}(F)$ is the union of the closures (w.r.t. Zarisky topology) of the regular zeroes of the T_i.

Wang (1993b) proposed a generalization of Wu's decompositions for affine varieties. This approach allows the resolution of quasi-algebraic systems. In this case $\mathbf{V}(F)$ is given as a (finite) union of regular zero sets of triangular systems (Definition 3.1). This method involves Wu's triangular sets but its process is different from Wu's one and seems to be more efficient.

Let us give an example to illustrate the difference between Wu, Lazard and Wang's way of solving, and Kalkbrener's one. We consider the system given by the following polynomials, where the ordered variables are $c_2 > s_2 > c_1 > s_1 > b > a$ and where the coefficients lie in the field of rational numbers:

$$\{c_1c_2 - s_1s_2 + c_1 - a, s_1c_2 + c_1s_2 + s_1 - b, c_1^2 + s_1^2 - 1, c_2^2 + s_2^2 - 1\}.$$

Our implementation of Lazard's method produces the decomposition $\{T_1, T_2, T_3\}$, where:

$$\begin{aligned} T_1 = \{&(4b^2 + 4a^2)s_1^2 + (-4b^3 - 4a^2b)s_1 + b^4 + 2a^2b^2 + a^4 - 4a^2,\\ &2ac_1 + 2bs_1 - b^2 - a^2,\\ &2as_2 + (2b^2 + 2a^2)s_1 - b^3 - a^2b,\\ &2c_2 - b^2 - a^2 + 2\},\\ T_2 = \{&a, 2s_1 - b, 4c_1^2 + b^2 - 4, s_2 - bc_1, 2c_2 - b^2 + 2\},\\ T_3 = \{&a, b, c_1^2 + s_1^2 - 1, s_2, c_2 + 1\}. \end{aligned}$$

What does this solution mean? In T_1, one may arbitrarily choose a and b once the product $a(b^2 + a^2)$ is not zero, and obtain successively the values of the indeterminates s_1, c_1, s_2, c_2. The triangular sets T_2 and T_3 describe the case $a = 0$. Note that in T_2, one may choose an arbitrary b whereas it is zero in T_3. So, where is the case $b^2 + a^2 = 0$? It is described by T_3. In fact, if we add this equation to the input system the computed decomposition is only $\{T_3\}$. Now, our implementation of Kalkbrener's algorithm produces the decomposition $\{C\}$, where:

$$\begin{aligned} C = \{&(4b^2 + 4a^2)s_1^2 + (-4b^3 - 4a^2b)s_1 + b^4 + 2a^2b^2 + a^4 - 4a^2,\\ &2ac_1 + 2bs_1 - b^2 - a^2,\\ &s_2 - bc_1 + as_1,\\ &s_1c_2 + bc_1^2 - as_1c_1 + s_1 - b\}. \end{aligned}$$

In this case a point is a solution of the input system if it lies in the closure of the set of the regular zeroes of C. Although C and T_1 are different, they have the same regular zero sets and the closure of these sets contains the regular zeroes of the previous T_2 and T_3. Thus Kalkbrener's output is simpler, however, further computations are needed to describe the zeroes satisfying $a(b^2 + a^2) = 0$.

In the conclusion of Kalkbrener (1993) the author writes: "a comparison with the algorithms of Ritt, Wu and Lazard seems to be interesting. In Wang (1993b) the author concludes: "A systematic analysis and comparison among them (the elimination methods of Lazard, Kalkbrener and Wang) both theoretically and practically remain interesting for future work".

The aim of this paper is to compare from a practical point of view the methods of Wu (1987), Lazard (1991), Kalkbrener (1991) and Wang (1993b). We realized a unified implementation in the AXIOM computer algebra system (Jenks and Sutor, 1992; Broadbery *et al.*, 1994) in order to compare these four methods. In Section 4, we discuss the matter of comparing the capabilities of different algorithms. Let us mention here that a crucial point is that their implementations need to share the same polynomial arithmetic and data structures.

The complexity of these algorithms, based on pseudo-division, is not known and to determine it is still a challenging task. Anyway, theoretical complexity considerations are not sufficient for the development of Computer Algebra; it is crucial to evaluate the possible efficiency of the algorithms on an experimental level.

A few papers report on the implementation of some methods for computing triangular decompositions of algebraic systems. Moreover, the examples which are presented in these papers differ from one to the other and the characteristics of the computer are generally not very detailed. Thus, the capabilities of the different algorithms do not appear clearly. Besides, it is important to focus on the properties of the output (representation, size, legibility, ...) since a triangular decomposition of a given polynomial system is not uniquely defined and two distinct implementations of the same method may produce distinct outputs on the same example.

Let us recall that an experimental comparison between Wu's method and Wang's triangular-series-based method is reported in Wang (1996). It appears that Wang's method is more efficient than Wu's characteristic set method. This extensive comparison is performed with the three notions of reduction proposed in Wu (1987). Another notion of reduction is presented here (Definition 2.1). Our implementation of Wu's algorithm is realized with this new notion and confirm the experimental results of Wang (1996).

Obviously, the methods considered in this paper can also be applied to a (lexicographical) Gröbner basis as input. It happens sometimes that this is the only way to obtain a result in a human time. However, we restrict ourselves to methods which accept any set of equations as input and do not compute Gröbner bases explicitly.[†] This choice was influenced by complexity reasons: Gröbner bases have double exponential behavior in worst cases—see Mayr and Meyer (1982) and Huynh (1986)—and direct triangular methods may simply have exponential complexity as indicated by Gallo and Mishra (1990). This is a strong reason for solving polynomial systems by means of methods which do not rely on explicit Gröbner bases computations.

Thus our comparative implementation excludes methods like *Lextriangular* (Lazard, 1992) or the approach of Möller (1993), which is generalized in Gräbe (1995).

We also specify that polynomial factorization (into irreducibles) is not a necessary tool in the four methods of our comparison. It is true that this technique is exploited in some implementations since it may discover some splits and thus make the decomposition easier to obtain. However, it is not a general rule, and systematic factorizations may also

[†]It may happen that a triangular decomposition consists of a single triangular set which is a lexicographical Gröbner basis.

be costly in difficult problems (and the AXIOM factorizer is not very efficient). After some preliminary tests it appeared that the use of factorization did not really modify the differences between the four methods of our comparison. Hence, we implemented the methods without using this technique.

Finally, we would like to remark that since the beginning of our work some improvements on the methods described in this paper, together with new triangular decomposition algorithms, have been proposed (Moreno Maza, 1998; Wang, 1998; Aubry, 1999).

The paper is organized as follows. Section 2 presents the general notions used in the paper. In Section 3, we review the main features of the methods and the properties of their decompositions. We also propose an inductive version of Wang's method. In Section 4, we develop the requirements which have guided our work and we present our implementation. Section 5 reports on some experimental data on a set of test examples. Most of them can be found in the data base of the research project PoSSo (European Commission, 1992) and are available in `ftp://www-calfor.lip6.fr/pub/papers/TriangularSets`. We investigate the computed decompositions for some relevant examples and point out some remarks suggested by our experimentations. In Section 6, we take into account some experimental results of the literature, which may be related to other approaches for computing triangular decompositions of algebraic systems. Finally, we present some conclusions about our work.

2. Definitions

The terminology and the notations concerning polynomials and triangular sets are exactly the same as in Aubry *et al.* (1999). We do not recall them in this paper except for the notions related to polynomial reduction. The four methods rely on pseudo-division and they use various notions of polynomial *reduction* related to this operation (Notation 2.1).

In the case of Wu's method, choosing one notion of reduction rather than another may strongly affect the efficiency (run time) and the legibility of the output. Wu considered the notion of *reduced* polynomial and the weaker notion of *head reduced* polynomial. He also remarked that his method was more efficient without any notion of reduction, but the termination of the computations was not guaranteed in this case. Here we introduce the new notion of *initially reduced* polynomial. This notion is weaker than the previous ones. However, it still insures the termination of Wu's algorithm and generally speeds up the computations, so we use it in our implementation.

We also review the important concept of normalization (Lazard, 1991, 1992) which strengthens the notion of initially reduced polynomial. This concept is also related to the notion of regular chain (Definition 2.2) since a normalized triangular set (Definition 2.1) is a regular chain.

Notation 2.1. Let $p, q \in \mathbf{P}_n$ with $q \notin \mathbf{k}$. We denote by $\mathsf{prem}(p, q)$ and $\mathsf{pquo}(p, q)$ the pseudo-remainder and the pseudo-quotient of p by q when interpreting them as univariate in $\mathsf{mvar}(q)$. Let us denote by $\mathsf{iter}(p)$ the triangular set of $\mathbf{P}_n$ recursively defined as follows: if $p \in \mathbf{k}$ then $\mathsf{iter}(p) = \emptyset$ otherwise $\mathsf{iter}(p) = \{p\} \cup \mathsf{iter}(\mathsf{init}(p))$. The elements of $\mathsf{iter}(p)$ are called the *iterated initials* of p. For instance, with $p = (x_2x_3 + x_2^3)x_4 - 2x_1$ we have $\mathsf{iter}(p) = \{x_2, x_2x_3 + x_2^3, p\}$. We say that p is *reduced w.r.t.* q if $\mathsf{deg}(p, \mathsf{mvar}(q)) < \mathsf{mdeg}(q)$. We say that p is *initially reduced w.r.t.* q if for every $h \in \mathsf{iter}(p)$ the condition $\mathsf{mvar}(h) = \mathsf{mvar}(q)$ implies $\mathsf{mdeg}(h) < \mathsf{mdeg}(q)$. Note that if p is reduced w.r.t. q then p is obviously initially reduced w.r.t. q. We say that p is *normalized w.r.t.* q if no polynomial in $\mathsf{iter}(p)$ has the same main variable as q. Thus, if p is normalized w.r.t. q, then p is initially reduced w.r.t. q.

Example 2.1. Let $p = (x_2x_3 + x_2^3)x_4 - 2x_1$. The polynomial p is normalized (but not reduced) w.r.t. the polynomial $q = x_1$ since no element in $\mathsf{iter}(p)$ has main variable x_1. If $q = x_1x_2^2 - 3$, then p is initially reduced w.r.t. q. Indeed, there exists a polynomial $h = x_2$ in $\mathsf{iter}(p)$ such that $\mathsf{mvar}(h) = \mathsf{mvar}(q)$ but $\mathsf{mdeg}(h) < \mathsf{mdeg}(q)$. However, p is neither normalized nor reduced w.r.t. q. If $q = x_3^2 - 2x_1x_2x_3 + 3x_1$, then p is reduced, but not normalized, w.r.t. q.

Definition 2.1. A triangular set T of $\mathbf{P}_n$ is called *reduced* (resp. *initially reduced*, *normalized*) if for every $v \in \mathsf{algVar}(T)$, the polynomial T_v is reduced (resp. initially reduced, normalized) w.r.t. each polynomial in T_v^-.

Example 2.2. Assume $n \geq 4$. Let $p_1 = x_1x_2^2 - 3$, $p_2 = x_3^2 - 2x_1x_2x_3 + 3x_1$ and $p_3 = (x_2x_3 + x_2^3)x_4 - 2x_1$. The subset $T_1 = \{p_1, p_2, p_3\}$ of $\mathbf{P}_n$ is an initially reduced triangular set, but it is neither reduced nor normalized. Let ${p_2}' = x_3 - 2x_1x_2$. The triangular set $T_2 = \{p_1, {p_2}', p_3\}$ is not initially reduced since the degree of the initial of p_3 w.r.t. x_3 is not smaller than the main degree of ${p_2}'$. Let ${p_3}' = 3(x_2 + 2x_1)x_4 - 2x_1$. The triangular set $T_3 = \{p_1, {p_2}', {p_3}'\}$ is reduced but not normalized. Finally, let ${p_3}'' = (12x_1^3 + 9)x_4 + 2x_1^2x_2 - 4x_1^3$. The set $T_4 = \{p_1, {p_2}', {p_3}''\}$ is a normalized triangular set.

Notation 2.2. For $i = 0, \ldots, n$ the saturated ideal of $T \cap \mathbf{P}_i$ in $\mathbf{P}_i$ is denoted by $\mathsf{sat}_i(T)$.

General triangular sets present some inconveniences. For instance, the affine variety associated with a triangular set may be empty (see the example in Remark 3.2). In order to guarantee better properties, the definition of triangular sets has been strengthened by several authors. Regular chains are particular triangular sets defined in Yang and Zhang (1994) and Kalkbrener (1991). For a regular chain T the set $\mathbf{W}(T)$ of its regular zeroes is not empty. By using the results of Section 4 in Aubry *et al.* (1999) we can simply define regular chains as follows:

Definition 2.2. A triangular set T in $\mathbf{P}_n$ is a *regular chain* if for each variable $x_i \in \mathsf{algVar}(T)$ the polynomial $\mathsf{init}(T_{x_i})$ does not belong to any prime ideal associated with $\sqrt{\mathsf{sat}_{i-1}(T_{x_i}^-)}$.

Example 2.3. Let p_1, p_2, p_3 and T_1 be as in Example 2.2. First, remark that $\{p_1, p_2\}$ is obviously a regular chain. Let $\mathcal{I}$ be the ideal generated by $\{p_1, p_2\}$ in $\mathbf{P}_3$. In fact, the ideal $\mathcal{I}$ is primary and it is the saturated ideal of the triangular set $\{p_1, p_2\}$. One can check that $\mathsf{init}(p_3)$ does not lie in the radical of $\mathcal{I}$, hence T_1 is a regular chain. Now, if $T_1' = \{p_1', p_2, p_3\}$ with $p_1' = x_2p_1$, then T_1' is not a regular chain. Indeed, let $\mathcal{I}'$ be the ideal generated by $\{p_1', p_2\}$ in $\mathbf{P}_3$. We have the following primary decomposition: $\mathcal{I}' = \mathcal{I} \cap \langle x_2, x_3^2 + 3x_1\rangle$, and it is clear that $\mathsf{init}(p_3)$ lies in the radical of $\langle x_2, x_3^2 + 3x_1\rangle$.

We now recall the definition of Lazard sets (Lazard, 1991).

NOTATION 2.3. Let T be a triangular set of $\mathbf{P}_n$ and $i \in \{0, \ldots, n\}$. From now on, we define $\mathbf{A}_i = \mathbf{fr}(\mathbf{P}_i/\mathsf{sat}_i(T))$. We denote by F_i the algebra homomorphism from $\mathbf{P}_{i+1}$ onto $\mathbf{A}_i[x_{i+1}]$ such that $F_i(x_{i+1}) = x_{i+1}$ and such that for any polynomial $p \in \mathbf{P}_i$ the value $F_i(p)$ is the canonical image of p in $\mathbf{A}_i$.

DEFINITION 2.3. Let $T \subseteq \mathbf{P}_n$ be a triangular set. For each variable $x_i \in \mathsf{algVar}(T)$ we put $t_i = T_{x_i}$ and denote by t_i' the derivative of t_i w.r.t. x_i. We say that T is a *Lazard set* if T is normalized, and if for each variable $x_i \in \mathsf{algVar}(T)$ we have

(i) [square-free] the ideal generated by $F_{i-1}(t_i)$ and $F_{i-1}(t_i')$ in $\mathbf{A}_{i-1}[x_i]$ is the unit ideal,
(ii) [primitive] for every $j = 1, \ldots, i-1$, the coefficients of t_i, interpreted as a multivariate polynomial in $(\mathbf{A}_{j-1}[x_j])[x_{j+1}, \ldots, x_i]$ generate the unit ideal of $\mathbf{A}_{j-1}[x_j]$.

REMARK 2.1. If T is a Lazard set of $\mathbf{P}_n$, then for each $i \in \{0, \ldots, n\}$, the ring $\mathbf{A}_i$ is a finite product of fields. See Moreno Maza (1997) for a proof of this statement. Now, let $\mathbf{A}_i = \mathbf{K}_{i,1} \times \cdots \times \mathbf{K}_{i,r_i}$, where the $\mathbf{K}_{i,j}$ are fields. Condition (i) means that for $\ell = 1, \ldots, r_{i-1}$ the image of t_i modulo $\mathbf{K}_{i-1,\ell}$ is square-free. Condition (ii) can be viewed as follows: for each $j = 1, \ldots, i-1$, and for $\ell = 1, \ldots, r_{j-1}$, the image of t_i in $(\mathbf{K}_{j-1,\ell}[x_j])[x_{j+1}, \ldots, x_i]$ is a primitive multivariate polynomial.

EXAMPLE 2.4. Let $T = \{x_1x_2^2 - 3, x_3^2 - 2x_1x_2x_3 + 3x_1\}$. Here $t_3 = x_3^2 - 2x_1x_2x_3 + 3x_1$ and $t_3' = 2x_3 - 2x_1x_2$. We have $\mathbf{A}_2 = \mathbf{k}(x_1)[x_2]/\langle x_1x_2^2 - 3\rangle$. Since $4t_3 \equiv {t_3'}^2$ modulo the ideal $\langle x_1x_2^2 - 3\rangle$, the polynomial $F_2(t_3)$ lies in the ideal generated by $F_2(t_3')$ in $\mathbf{A}_2[x_3]$. Thus, T is not a Lazard set. In fact, the polynomial t_3' is the square-free form of t_3 if they are both interpreted in $\mathbf{A}_2[x_3]$ and $\{x_1x_2^2 - 3, t_3'\}$ is a Lazard set.

3. Methods

This section summarizes the four methods. We recall the specifications of each method together with the properties of the decompositions that they compute. A detailed review of these methods could not take place here. The reader may refer to the original papers.

We present the first method proposed in Wang (1993b). In fact, we suggest here a recursive presentation of this method. Note that the decompositions of Wang's algorithm are different from the other ones in the sense that they are not triangular sets but *fine triangular quasi-algebraic systems* (see Definition 3.1).

The basic ideas of Wu's method are given but for more details one can refer to Wu (1987) or Wang (1991).

We recall the main features of the methods of Lazard (1991) and Kalkbrener (1991) which both involve gcd computations over towers of simple extensions.

Our implementation of Lazard's method is based on an algorithm for gcd computations of univariate polynomials with coefficients in a separable tower of simple extensions (Moreno Maza, 1997). The idea is a generalization of the one of Moreno Maza and Rioboo (1995). This algorithm is rather technical and could not be outlined here.

We use the AXIOM programming language for describing the algorithms presented in this section.

Notation 3.1. Let $p \in \mathbf{P}_n$ be a polynomial and $T \subseteq \mathbf{P}_n$ be a triangular set. There exists a polynomial $r \in \mathbf{P}_n$, initially reduced w.r.t. T, and a product s of the initials of T, such that $sp - r$ lies in the ideal generated by T. We define $\mathsf{iRed}(p, T) = r$. Now, for $F \subseteq \mathbf{P}_n$ we set $\mathsf{iRed}(F, T) = \{\mathsf{iRed}(p, T)\ , p \in F\}$. Finally, we define $\mathsf{prem}(F, T) = \{\mathsf{prem}(p, T)\ , p \in F\}$.

3.1. Wang's method

Wang's method computes a finite family $\{(T_1, Q_1), \ldots, (T_r, Q_r)\}$ of fine triangular q.a.s. (see Definition 3.1) such that

$$\mathbf{V}(F) = \bigcup_{i=1}^{r} \mathbf{Z}(T_i, Q_i).$$

Such a decomposition is produced by $\mathsf{triangulation}(F, \emptyset, \emptyset)$ (see Theorem 3.1). There is no reason for a fine triangular system produced by the method of Wang described below (called *elimination without projection* in Wang, 1993b) to be necessarily consistent. However, maybe due to our optimizations, we never encountered an inconsistent fine triangular system during our experiences. Note that Wang also proposes a method called *elimination with projection* to produce necessarily consistent outputs.

Definition 3.1. Every couple $\Sigma = (P, Q)$, where P and Q are two finite subsets of $\mathbf{P}_n$, is called a *quasi-algebraic system* in $\mathbf{P}_n$ (*q.a.s.* for short). Let $\Sigma = (P, Q)$ be a q.a.s. in $\mathbf{P}_n$. The q.a.s. Σ is called *triangular* if P is a triangular set of $\mathbf{P}_n$. If $Q \neq \emptyset$, then we denote by $h(\Sigma)$ the product of the elements of Q, otherwise we define $h(\Sigma) = 1$. We call a *zero* of Σ every element of the subset of $\mathbf{K}^n$ denoted by $\mathbf{Z}(\Sigma)$ and defined by:

$$\mathbf{Z}(\Sigma) = \mathbf{V}(P) \setminus \mathbf{V}(h(\Sigma)).$$

The q.a.s. Σ is called *inconsistent* if $\mathbf{Z}(\Sigma) = \emptyset$, otherwise it is called *consistent.* Finally, a triangular q.a.s. $\Sigma = (T, Q)$ is called *fine* if $\mathbf{V}(h(T)) \cap \mathbf{Z}(\Sigma) = \emptyset$ and $0 \notin \mathsf{prem}(Q, T)$ where $h(T)$ is the product of the initials of T.

Let $\Sigma = (P, Q)$ be a q.a.s. in $\mathbf{P}_n$. From now on we assume that $P \not\subseteq \mathbf{k}$. Let x_i be the greatest variable occurring in P, denoted by $\mathsf{mvar}(P)$. The algorithm $\mathsf{elimination}(x_i, P, Q)$ presented below (Proposition 3.1) splits the q.a.s. Σ into several q.a.s. which contain at most one equation with x_i as main variable (see Definition 3.2). Its proof is based on the following Lemma 3.1 (Wang, 1993a) and Lemma 3.2 (which is a practical remark whose proof is straightforward).

Definition 3.2. Let $1 \leq i \leq n$ and $\Sigma = (P, Q)$ be a q.a.s. in $\mathbf{P}_n$ such that $P \subseteq \mathbf{P}_i$ and $Q \subseteq \mathbf{P}_n$. We call *elimination* of the variable x_i in Σ a set Λ of triplets (P_k, Q_k, τ_k) such that P_k, Q_k and τ_k are finite subsets of $\mathbf{P}_n$ which satisfy the following conditions:

(i) $P_k \neq \emptyset \Rightarrow \mathsf{mvar}(P_k) < x_i$,
(ii) $\tau_k \neq \emptyset\ \Rightarrow (\exists t \in \mathbf{P}_i \setminus \mathbf{P}_{i-1}) \mid \tau_k = \{t\}$,
(iii) $\mathbf{Z}(P, Q) = \bigcup_{(P_j, Q_j, \tau_j) \in \Lambda} \mathbf{Z}(P_j \cup \{\tau_j\}, Q_j)$.

Lemma 3.1. *Let f be a non-constant polynomial in $\mathbf{P}_n$ and (P, Q) a q.a.s. in $\mathbf{P}_n$. Then*

$$\mathbf{Z}(P \cup \{f\}, Q) = \mathbf{Z}(\mathsf{prem}(P, f) \cup \{f\}, Q \cup \{\mathsf{init}(f)\}) \cup \mathbf{Z}(P \cup \{\mathsf{init}(f), \mathsf{tail}(f)\}, Q).$$

Lemma 3.2. *Let (P, Q) be a q.a.s. in $\mathbf{P}_n$ and $f \in \mathbf{P}_n \setminus \mathbf{k}$. Then*

$$\mathsf{init}(f) \in Q \Rightarrow \mathbf{Z}(P \cup \{f\}, Q) = \mathbf{Z}(P \cup \{f\}, \mathsf{prem}(Q, f)).$$

Proposition 3.1. *Let v be a variable in $\{x_1, \ldots, x_n\}$ and (P, Q) a q.a.s. in $\mathbf{P}_n$ such that $\mathsf{mvar}(P) \leq v$. Then the algorithm $\mathsf{elimination}(v, P, Q)$ given below, computes an elimination of the variable v in the q.a.s. (P, Q). In particular, if the output of the algorithm is the empty set, then $\mathbf{Z}(P, Q) = \emptyset$.*

```
• elimination($v, P, Q$) ==
      $P := P \setminus \{0\}$
      $(0 \in Q)$ or $(P \cap \mathbf{k} \neq \emptyset)$ => return($\{\}$)
      $P_v^- := \{p \in P \mid \mathsf{mvar}(p) < v\}$
      $P_v := P \setminus P_v^-$
      $P_v = \emptyset$ => return ( $\{(P, Q, \emptyset)\}$ )
      $f$ := a polynomial in $P_v$ with minimal degree in $v$
      $P_1 := (P_v \setminus \{f\}) \cup \{\mathsf{init}(f), \mathsf{tail}(f)\} \cup P_v^-$
      $Q_2 := Q \cup \{\mathsf{init}(f)\}$
      empty? $(P_v \setminus \{f\})$ => return ($\{(P_v^-, \mathsf{prem}(Q_2, f), \{f\})\} \cup \mathsf{elimination}(v, P_1, Q)$)
      $P_2 := \mathsf{prem}(P_v \setminus \{f\}, f) \cup \{f\} \cup P_v^-$
      return ($\mathsf{elimination}(v, P_2, Q_2) \cup \mathsf{elimination}(v, P_1, Q)$)
```

Proof. Let us define $s(P) = \sum_{p \in P \setminus \{0\}} \mathsf{deg}(p, v)$. We will prove termination and correctness by induction on $s(P)$. If a constant occurs in P or if $0 \in Q$, the result is obvious. Otherwise, if $s(P) = 0$, then $P_v = \emptyset$ and the algorithm terminates. The correctness is obvious. Now we assume that $s(P) > 0$, i.e. P_v is not empty. First, we remark that $s(P_1) < s(P)$ since $\mathsf{deg}(\mathsf{init}(f), v) = 0$ and $\mathsf{deg}(\mathsf{tail}(f), v) < \mathsf{deg}(f, v)$. Two cases arise. First, assume that P_v consists of the single polynomial f. By induction, $\mathsf{elimination}(v, P_1, Q)$ terminates and is correct. Thus $\mathsf{elimination}(v, P, Q)$ terminates and correction follows from Lemmas 3.1 and 3.2. Now assume that $P_v \setminus \{f\} \neq \emptyset$. Let us denote by P' the set $P_v \setminus \{f\}$. For any $p \in P'$, we have $\mathsf{deg}(\mathsf{prem}(p, f), v) < \mathsf{deg}(f, v) \leq \mathsf{deg}(p, v)$. Since P' is not empty, we obtain $s(\mathsf{prem}(P', f)) < s(P')$, and consequently $s(P_2) < s(P)$. Then termination and correctness follow by application of Lemma 3.1 and induction hypothesis.

By repeated use of the algorithm elimination with v decreasing, we easily obtain a triangulation of any q.a.s. with the algorithm triangulation presented below.

Theorem 3.1. *Let $1 \leq i \leq n$ and (P, Q) be a q.a.s. in $\mathbf{P}_n$ such that $P \subseteq \mathbf{P}_i$. Let T a triangular set of $\mathbf{P}_n$ such that $T \cap \mathbf{P}_i = \emptyset$. Then the following algorithm $\mathsf{triangulation}(P, Q, T)$ computes a finite family $\{(T_1, Q_1), \ldots, (T_r, Q_r)\}$ of triangular q.a.s. such that*

$$\mathbf{Z}(P \cup T, Q) = \bigcup_{k=1}^{r} \mathbf{Z}(T_k, Q_k).$$

```
• triangulation$(P, Q, T)$ ==
      $P := P \setminus \{0\}$
      $(0 \in Q)$ **or** $(P \cap \mathbf{k} \neq \emptyset)$ => **return** $(\{\})$
      **empty?** $P$ => **return** $(\{(T, Q)\})$
      $v :=$ mvar$(P)$
      $\Lambda :=$ elimination$(v, P, Q)$
      **return** $(\bigcup_{(P_j, Q_j, \tau_j) \in \Lambda}$ triangulation$(P_j, Q_j, \tau_j \cup T))$
```

Proof. The proof of the algorithm is obtained by induction on the smallest integer $i(P)$ such that $P \subseteq \mathbf{P}_{i(P)}$. For $i(P) = 0$, i.e. $P \subseteq \mathbf{k}$, the result is obvious. Now assume that $i(P) > 0$. We do not need to consider the case $0 \in Q$, the case $P \cap \mathbf{k} \neq \emptyset$, and the case $\Lambda = \emptyset$, for which the termination and the correctness are obvious. Then, due to the specifications of the algorithm elimination, we obtain

$$\mathbf{Z}(P \cup T, Q) = \mathbf{Z}(P, Q) \cap \mathbf{V}(T) = \bigcup_{(P_j, Q_j, \tau_j) \in \Lambda} \mathbf{Z}(P_j \cup (\{\tau_j\} \cup T), Q_j).$$

Let us denote $T_j = \{\tau_j\} \cup T$. The triplets (P_j, Q_j, T_j) satisfy the input conditions of triangulation. Since $i(P_j) < i(P)$, the result follows from the induction hypothesis.

Remark 3.1. It is easy to check that the quasi-algebraic systems produced by the algorithm triangulation$(F, \emptyset, \emptyset)$ are fine.

3.2. Wu's method

Wu used Ritt's work to provide an algorithm for solving systems of algebraic equations by means of triangular sets which only requires pseudo-remainder computations (i.e. no factorizations are needed). Wu's process is based on a procedure called CHRST-REM, see p. 3 in Wu (1987). Given a finite subset F of $\mathbf{P}_n$, this procedure computes a *characteristic set* T of a finite subset G of $\mathbf{P}_n$ such that F and G generate the same ideal in $\mathbf{P}_n$. It involves Ritt ordering for (initially) reduced triangular sets of $\mathbf{P}_n$. For more details about Ritt ordering and characteristic sets the reader may consult Ritt (1932, 1966) and Aubry *et al.* (1999).

Definition 3.3. Let F be a non-empty finite subset of non-constant polynomials in $\mathbf{P}_n$. We call *basic set* of F a subset B of F such that B is a minimal element for Ritt ordering among the family of initially reduced triangular sets contained in F.

It is easy to compute a basic set of F. Let us denote by basicSet(F) the result of an algorithm which computes a basic set of F. Now the following operation charSet(F) either returns the set $C = \{1\}$ or computes an initially reduced triangular set C. In both cases C is called a *Wu characteristic set* of F and satisfies the following:

(i) $\overline{\mathbf{W}(C)} \subseteq \mathbf{V}(F) \subseteq \mathbf{V}(C)$,
(ii) $\mathbf{V}(F) = \mathbf{W}(C) \cup \bigcup_{p \in C} \mathbf{V}(F \cup \{\text{init}(p)\})$.

```
charSet$(F)$ ==
      $R := F \setminus \{0\}$
      $Q := \emptyset$
```

```
$C := \emptyset$
while $(R \neq \emptyset)$ and $(R \cap \mathbf{k} = \emptyset)$ repeat
    $Q := Q \cup R \cup C$
    $C := \mathsf{basicSet}(Q)$
    $Q := Q \setminus C$
    $R := \mathsf{iRed}(Q, C) \setminus \{0\}$
$R \cap \mathbf{k} \neq \emptyset$ => return$(\{1\})$
return $C$
```

Remark 3.2. The set C obtained by the above algorithm may be a fine triangular set (i.e. none of its initials reduces to 0 w.r.t. C) of F even if F generates the unit ideal of $\mathbf{P}_n$. For instance, choose $F = \{x_1^2 - x_1, x_1x_2 - 1, (x_1 - 1)x_3 + x_2\}$. In this case the algorithm $\mathsf{charSet}(F)$ returns F itself, but we have $\langle F \rangle = \langle 1 \rangle$.

Let $p \in C$. We remark that $\mathbf{V}(F \cup \{\mathsf{init}(p)\}) = \mathbf{V}(F \cup C \cup \{\mathsf{init}(p)\})$. Moreover, the set $C \cup \{\mathsf{init}(p)\}$ has obviously a basic set which is smaller than C w.r.t. Ritt ordering. Therefore, by computing $\mathsf{charSet}(F \cup C \cup \{\mathsf{init}(p)\})$, we obtain a smaller characteristic set than C w.r.t. Ritt ordering. Now one can easily check that every strictly decreasing chain of triangular sets for Ritt ordering is finite. Thus, by virtue of the above formula (ii), repeated calls to the algorithm $\mathsf{charSet}$ allow the computation of a finite family $\{T_1, \dots, T_r\}$ of initially reduced triangular sets such that

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

Remark 3.3. Note that not only some components $\mathbf{W}(T_i)$ of Wu's decomposition may be empty (as shown in Remark 3.2), but also the algorithm may produce superfluous components in the following sense: for some $i \in \{1, \dots, r\}$ such that $\mathbf{W}(T_i) \neq \emptyset$ there exists $j \in \{1, \dots, r\}$ with $i \neq j$ such that $\mathbf{W}(T_i)$ is contained in $\mathbf{W}(T_j)$.

3.3. Lazard's method

Let i be in the range $0 \dots n-1$. Let $T \subseteq \mathbf{P}_i \setminus \mathbf{P}_{i-1}$ be a Lazard set. We saw in Remark 2.1 that $\mathbf{A}_i = \mathbf{fr}(\mathbf{P}_i/\mathsf{sat}_i(T))$ is a product of fields. We put $\mathbf{A}_i = \mathbf{k}_1 \times \cdots \times \mathbf{k}_m$, where the $\mathbf{k}_\ell$ are fields. The main tool in Lazard's method is the computation of gcd in $\mathbf{A}_i[x_{i+1}]$. Let $p, q \in \mathbf{A}_i[x_{i+1}]$. To compute a gcd of p and q one may apply a standard algorithm in each $\mathbf{k}_\ell[x_{i+1}]$. But in practice the $\mathbf{k}_\ell$ are not known. So a variation of the *subresultant PRS algorithm*, proposed in Moreno Maza and Rioboo (1995), is performed in the ring $\mathbf{A}_i[x_{i+1}]$. As in Della Dora *et al.* (1985), Kalkbrener (1991) and Gómez-Díaz (1994), computations may be split when a zero-divisor is discovered. Full details of this gcd algorithm and the implementation of Lazard's method appear in Moreno Maza (1997).

The main procedure of Lazard's method is called $\mathsf{intersect}$. Given $T \subseteq \mathbf{P}_n$ and $p \in \mathbf{P}_n$ the operation $\mathsf{intersect}(p, T)$ returns a finite family of Lazard sets $\{S_1, \dots, S_\ell\}$ such that

$$\mathbf{V}(p) \cap \mathbf{W}(T) \subseteq \cup_1^\ell \mathbf{W}(S_i) \subseteq \overline{\mathbf{V}(p) \cap \mathbf{W}(T)}.$$

Given $\{T_1, \ldots, T_s\}$, a finite family of Lazard sets, we define $\mathsf{intersect}(p, \{T_1, \ldots, T_s\})$ as the union of the $\mathsf{intersect}(p, T_i)$. Then, given a finite subset $F = \{f_1, \ldots, f_m\}$ of $\mathbf{P}_n$ we define $\mathsf{intersect}(F, T) = \mathsf{intersect}(f_1, \mathsf{intersect}(\ldots, \mathsf{intersect}(f_m, T)))$. Thus, $\mathsf{intersect}(F, \emptyset)$ produces a finite family of Lazard sets $\{S_1, \ldots, S_l\}$ such that

$$\mathbf{V}(F) = \bigcup_{i=1}^{\ell} \mathbf{W}(S_i).$$

Lazard's decompositions are *non-redundant* in the following sense:

$$\bigcup_{j \neq i} \mathbf{W}(S_j) \neq \bigcup_{j} \mathbf{W}(S_j).$$

We will not describe here how to produce non-redundant decompositions. For this purpose, see Moreno Maza (1997). The operation $\mathsf{intersect}(p, T)$ proceeds in the following way. Let us recall that $F_n(p)$ is the image of p in $\mathbf{fr}(\mathbf{P}_n/\mathsf{sat}(T))$ (see Notation 2.3). Note also that an operation is available in our implementation to compute $F_n(p)$.

(l_1) If p is normalized w.r.t. T, then go to step (l_2) with $r = p$ else go to the next step.
($l_1{}'$) If p is not normalized w.r.t. T, compute two polynomials $q, r \in \mathbf{P}_n$ such that r is normalized w.r.t. T and $F_n(pq - r) = 0$ and $F_n(p) = 0 \iff F_n(r) = 0$. Polynomials q and r are computed by means of an extended (i.e. with Bezout coefficients) version of the gcd algorithm outlined above. Here the computations may be split, in particular if $F_n(p)$ is a zero-divisor. The polynomial r is also denoted by $\mathsf{normalize}(p, T)$. Now, go to the next step.
(l_2) If $r = 0$, then return $\{T\}$ and exit. If $r \neq 0$ and $r \in \mathbf{k}$, then return $\{\,\}$ and exit. If $r \notin \mathbf{k}$, then go to the next step.
(l_3) Return $\mathsf{intersect}(\mathsf{tail}(r), \mathsf{intersect}(\mathsf{init}(r), T))$ and go to the next step.
(l_4) Remove the content of r viewed as univariate in $\mathsf{mvar}(r)$ and go to the next step.
(l_5) If $T^-_{\mathsf{mvar}(r)} \cup \{r\}$ is a square-free regular chain (that is, which satisfies the square-free condition of Definition 2.3), then go to step (l_7).
(l_6) Let $v = \mathsf{mvar}(r)$. Compute a gcd (normalized w.r.t. T^-_v) of r and its derivative w.r.t. v while interpreting their coefficients in the tower associated with T^-_v. Here computations may be split. Let g be this gcd. Replace r by $\mathsf{pquo}(r, g)$. Thus $T^-_v \cup \{r\}$ is now a square-free regular chain. Go to step (l_3).
(l_7) Let $v = \mathsf{mvar}(r)$. Define $T^+_v = \{t_k, \ldots, t_l\}$ with $\mathsf{mvar}(t_k) < \cdots < \mathsf{mvar}(t_l)$. Compute $\mathcal{D} = \mathsf{intersect}(t_l, \mathsf{intersect}(\ldots, \mathsf{intersect}(t_k, T^-_v \cup \{r\})))$. Then remove from $\mathcal{D}$ any triangular set U such that $\mathsf{normalize}(\mathsf{init}(t_i), U^-_{\mathsf{mvar}(t_i)}) = 0$ for some $i \in \{k, \ldots, l\}$. Now, go to the next and last step.
(l_8) Return $\mathsf{intersect}(p, \mathcal{D})$, where p is the input polynomial.

3.4. Kalkbrener's method

Kalkbrener's method is not so incremental as Lazard's one. It computes a finite family $\{T_1, \ldots, T_r\}$ of regular chains but deals rather with varieties than regular zero sets. The decomposition is such that

$$\mathbf{V}(F) = \bigcup_{i=1}^{r} \mathbf{V}(\mathsf{sat}(T_i)).$$

Thus, by Theorem 2.1 in Aubry (1999), we have

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

Moreover, each $\overline{\mathbf{W}(T_i)}$ is not empty. However, there may exist i_1 and i_2 such that $\overline{\mathbf{W}(T_{i_1})} \subseteq \overline{\mathbf{W}(T_{i_2})}$. Note that when $\mathbf{V}(F)$ is zero-dimensional then $\mathbf{V}(F) = \cup_{i=1}^{r}\mathbf{W}(T_i) = \cup_{i=1}^{r}\mathbf{V}(T_i)$.

We think that a good way to sketch this method is to give the algorithm for computing triangular decompositions together with the specifications of Kalkbrener's algorithm for computing gcd modulo regular chains (Kalkbrener, 1998). The notion of gcd is similar as in Lazard's method. Let $T \subseteq \mathbf{P}_i$ be a regular chain and $\mathbf{B}_i = \mathbf{fr}(\mathbf{P}_i/\sqrt{\mathsf{sat}_i(T)})$. Aubry (1999) shows that $\mathbf{B}_i$ is the following product of fields:

$$\mathbf{B}_i \simeq \mathbf{fr}(\mathbf{P}_n/\mathcal{P}_1) \times \cdots \times \mathbf{fr}(\mathbf{P}_n/\mathcal{P}_m),$$

where $\mathcal{P}_1, \ldots, \mathcal{P}_m$ are the prime ideal associated to $\mathsf{sat}_i(T)$. The algorithm ggcd of Kalkbrener (1993) computes gcd of polynomials in $B_i[x_{i+1}]$ by a dynamical process like the algorithms of Moreno Maza and Rioboo (1995) and Moreno Maza (1997). More details about the way of implementing efficiently these algorithms and extensions of Kalkbrener's method are given by Aubry (1999).

Notation 3.2. Let $\mathcal{P}$ be an ideal in $\mathbf{P}_{n-1}$ and $p \in \mathbf{P}_n$. We denote by $\bar{p}^{\mathcal{P}}$ the canonical image of p in $\mathbf{fr}(\mathbf{P}_{n-1}/\mathcal{P})[x_n]$.

Algorithm. $\mathsf{gcd}_n(C, F)$

Input: C a regular chain in $\mathbf{P}_{n-1}$ and F a finite subset of $\mathbf{P}_n$.
Output: $\{(C_1, g_1), \ldots, (C_s, g_s)\}$ where every C_k is a regular chain in $\mathbf{P}_{n-1}$ and every g_k is a polynomial in $\mathbf{P}_n$ such that

(i) $\bigcup_1^s, \mathsf{ap}(\sqrt{\mathsf{sat_{n-1}(C_k)}}) = \mathbf{ap}(\sqrt{\mathsf{sat_{n-1}(C)}})$
(ii) for all $\mathcal{P} \in \mathsf{ap}(\sqrt{\mathsf{sat}_{n-1}(C_k)})$,
1 $F = \emptyset \Rightarrow s = 1$ and $g_1 = 0$,
$F \neq \emptyset \Rightarrow \overline{g_k}^{\mathcal{P}}$ is the gcd of $\{\bar{f}^{\mathcal{P}}, f \in F\}$ for each k,
2 if $g_k \in \mathbf{k}$ and $\mathsf{mvar}(g_k) = x_n$ then $\mathsf{init}(g_k) \notin \mathcal{P}$,
if $g_k \in \mathbf{k}$ and $\mathsf{mvar}(g_k) < x_n$ then $g_k \notin \mathcal{P}$,
3 $g_k \in \langle \sqrt{\mathsf{sat}(C_k)} \cup F \rangle$.

Algorithm. $\mathsf{decompose}_n(F)$

Input: F a finite subset of $\mathbf{P}_n$.
Output: regular chains $T_1, \ldots, T_r$ of $\mathbf{P}_n$ such that $\sqrt{\langle F \rangle} = \cap_{j=1}^{r}\sqrt{\mathsf{sat}(T_j)}$

```
decompose_n(F) ==
    $F := F \backslash \{0\}$
    **empty?** $F$ => $\{\emptyset\}$
    $F \cap \mathbf{R} \neq \emptyset$ => $\{\ \}$
```

```
$\Theta := \emptyset$
$F' := F \cap \mathbf{P}_{n-1}$
$\Delta := \mathsf{decompose}_{n-1}(F')$
**for** $C \in \Delta$ **repeat**
    $\Gamma := \mathsf{gcd}_n(C, F \backslash F')$
    **for** $(C_j, g_j) \in \Gamma$ **repeat**
        $g_j = 0 => \Theta := \Theta \cup \{C_j\}$
        $\mathsf{mvar}(g_j) < x_n => \Theta := \Theta \cup \mathsf{decompose}_n(F \cup \{g_i\})$
        $\Theta := \Theta \cup \{C_j \cup \{g_j\}\} \cup \mathsf{decompose}_n(F \cup \mathsf{init}(g_j))$
**return** $\Theta$
```

4. Implementation

4.1. general requirements

In the introduction we specified why comparing methods for computing triangular decompositions is not an easy task. Recall that each of the algorithms studied in this paper has its own specifications and that a triangular decomposition of a polynomial system by a given method is not uniquely defined.

In order to realize a reasonable comparison we think that the following requirements should meet.

(1) The algorithms must be implemented and run with the same human, material and software conditions (using the same data structures and sub-routines).
(2) A process to check the correctness of the computed decompositions must be implemented.
(3) The experiments must not only focus on timings but also on the legibility of the outputs and their suitability for further computations.

The goal of the first requirement is to get as close as possible to the ideal situation where the differences between computations of triangular decompositions—for a given system—only depend on the corresponding algorithms. Thus our implementations of these methods need to use the same data structures and sub-routines. It is clear that even with the same hardware and software, experimental comparisons strongly depend on some implementation choices.

We thought that the AXIOM computer algebra system (Jenks and Sutor, 1992; Broadbery *et al.*, 1994), with its strongly typed and object-oriented language, is convenient to satisfy our first requirement. We defined categories corresponding to the different properties of triangular sets, packages and domains for the common data structures and sub-routines. Furthermore, AXIOM (version 1.2) is connected with GB, the very powerful Gröbner engine developed by Faugère (1994). This allowed us to run the non-trivial Gröbner basis computations that are required in order to satisfy our other two requirements.

We mentioned that two implementations of the same method for computing triangular decompositions may produce different outputs for a given polynomial system. Therefore it is difficult to be sure that a decomposition is correct. We concentrated on this problem rather than trying to produce very optimized implementations. We think that only checking *by hand* some computations (necessarily simple) produced by an implementation is not sufficient to make sure that this implementation is correct, especially for mixed-dimensional problems. For instance, we discovered a bug in the management of the elimination of redundant branches in our implementation of Wu's method by verifying the computed output on Liu's example. More generally, our checking process (described below) is a convenient debugging tool for our implementations.

This checking process

(i) has been intensively tested for more than a year,
(ii) is based on simple and well-known algorithms and
(iii) is implemented in a direct way in AXIOM as an top-level package of the GB software.

Thus it can be safely considered as reliable.

In our analysis of the computed solutions we also looked for other information than timings and correctness. Given a solution, we wanted to know if some of the computed triangular sets are inconsistent or if some quasi-components $\mathbf{W}(T_i)$ are contained in another quasi-component $\mathbf{W}(T_j)$ (or in the closure of another quasi-component). An overview of these facilities is given in the subsequent paragraph.

Some of the information extracted from our experiments could not be presented here and is reported together with all the computed outputs in Moreno Maza (1997).

4.2. DESCRIPTION OF THE IMPLEMENTATION

Each implementation of the four methods uses the same AXIOM domain for polynomials (with a sparse and recursive representation). Let us recall that an AXIOM category specifies the mathematical properties of the domains which belong to this category. It may also give default definitions for exported operations (eventually redefined in the domains of this category). The main categories and domains of our implementation are described below and their hierarchy is illustrated in Figure 1.

First we defined a category *PolynomialSetCategory* for finite subsets of $\mathbf{P}_n$. This category exports and implements operations on sets, ideals and varieties like $(I, J) \longmapsto I \cap J$ and $(I, p) \longmapsto I : p^\infty$, where $I, J \subseteq \mathbf{P}_n$ denote ideals and $p \in \mathbf{P}_n$ is a polynomial. We implemented these operations by means of Gröbner bases techniques (Cox *et al.*, 1992) in an AXIOM package using the connection between AXIOM and the powerful Gröbner engine GB.

Then we wrote a category for triangular sets of $\mathbf{P}_n$ named *TriangularSetCategory*. This category exports and implements basic operations like $(T, v) \longmapsto T_v$ and $(p, T) \longmapsto \mathsf{prem}(p, T)$ and $(p, T) \longmapsto \mathsf{iRed}(p, T)$ (Notation 3.1), where v is a variable and $T \subseteq \mathbf{P}_n$ is a triangular set. It also exports and implements more sophisticated operations like:

(i) $T \longmapsto \mathsf{sat}(T)$ which computes a Gröbner basis of the saturated ideal of T,
(ii) $(F \subseteq \mathbf{P}_n, \{T_1, \ldots, T_r \subseteq \mathbf{P}_n\}) \longmapsto \mathbf{V}(F) \overset{?}{=} \cup_i \overline{\mathbf{W}(T_i)}$ which tests if the variety $\mathbf{V}(F)$ is the union of the closures of the $\mathbf{W}(T_i)$.

We use the operations from the category *PolynomialSetCategory* that we mentioned above to perform these latter operations. That way we can check the consistency of a triangular set and the correctness of a triangular decomposition.

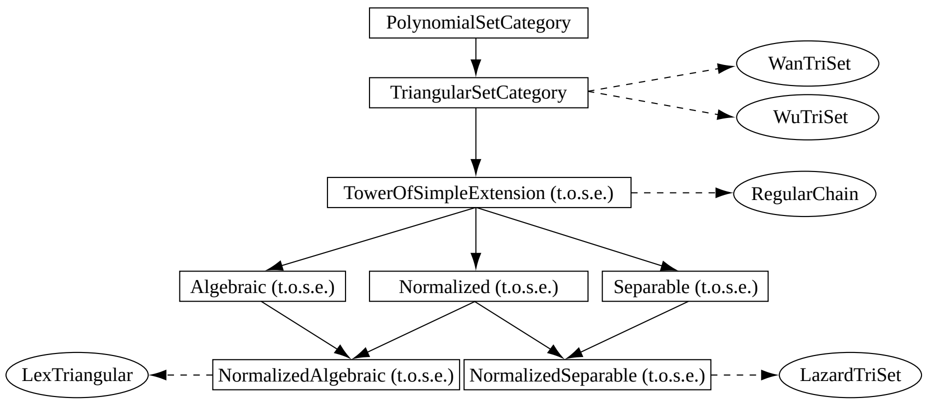

Figure 1. Categories and domains of the implementation

Moreover, the category *TriangularSetCategory* exports (but does not implement) an operation $F \subseteq \mathbf{P}_n \longmapsto \mathsf{zeroSetSplit}(F)$ which represents any method for solving polynomial system by means of triangular systems.

From the category of general triangular sets we derived a category for *towers of simple extensions* (t.o.s.e.) which corresponds to the properties of regular chains. It exports the associated map of a t.o.s.e. implemented with the operation $(p, T) \longmapsto F_n(p)$ (Notation 2.3). It also exports operations like $(p, T) \longmapsto$ is-$F_n(p)$-a-unit ?.

Finally, from the category of t.o.s.e. we derived three categories corresponding to particular properties.

(1) A category for the towers $T \subseteq \mathbf{P}_n$ such that $\mathsf{algVar}(T) = \{x_1, x_2, \ldots, x_n\}$, called *algebraic t.o.s.e.*,
(2) A category for the normalized towers called *normalized t.o.s.e.*,
(3) A category for the square-free towers (that is, which satisfy square-free condition of Definition 2.3) called *separable t.o.s.e.*.

Each method for computing triangular decompositions is an implementation of the operation $F \subseteq \mathbf{P}_n \longmapsto \mathsf{zeroSetSplit}(F)$ in an AXIOM domain of the suitable category. For instance, Kalkbrener's method is implemented in an domain which belongs to the category of t.o.s.e. and which is called *RegularChain* (see Figure 1). Note that the second author implemented the lexTriangular method (Lazard, 1992) by using the techniques described in Moreno Maza and Rioboo (1995). This algorithm is implemented in an AXIOM domain which belongs to both categories of normalized t.o.s.e. and algebraic t.o.s.e.

4.3. SOME TECHNIQUES USED IN THE IMPLEMENTATION

Before studying the implementation of each method, let us give some common techniques and optimizations.

We mentioned in the introduction that we do not use polynomial factorization into irreducibles. However, square-free factorization of multivariate polynomials has a lower cost. We use it at certain steps of each method, in particular in order to clean up the triangular sets in the outputs: this increases their legibility and helps to discover inconsistent components in Wu's method.

Another common technique is the use of what we call *pre-processing*: the idea is to perform some *inter-reductions* in a polynomial set Q before calling a procedure with Q as input. This is done, for instance, in Wu's method before calling charSet(Q). More precisely, polynomials in Q with constant initials are used to reduce the other polynomials in Q. This speeds up the computation of a characteristic set from Q. In the same way, in Wang's method, before running elimination(v,P,Q) we reduce the polynomials with main variable v by the polynomials in P_v^- with a constant initial. It appears that this also improves our implementation. For the methods of Kalkbrener and Lazard, a *pre-processing* is only performed with the input system F. In this case, for each main variable v of some polynomial in F, we choose a polynomial f_v in F_v, which is minimal w.r.t. Ritt and Wu ordering. Then, we use these polynomials f_v to reduce the other polynomials in F in the sense of Gröbner bases (i.e. we use the division algorithm in $\mathbf{P}_n$ as in Cox *et al.*, 1992, p. 59). In many examples (*Arnborg–Lazard, Gonnet, Hairer-2, Butcher*), this speeds up the computation of a triangular decomposition. Sometimes, this may also slow down the computations (*Gerdt, Liu–Li*). However, this *pre-processing* is used for both methods with all examples.

Concerning the case of Wu's method, most of the optimizations that we use appear in Wu (1987) and Wang (1992). Note that the removal of the redundant branches for this method is not easy since empty components may be produced. Thus, many heuristics are needed.

Several techniques are given in Wang (1993b) to improve the practical efficiency of the method. In particular, the author points out the removal of the redundant factors. By means of gcd computations and exact divisions, our implementation actually removes the redundant factors which appear among the set of inequations, or between equations and inequations. Another technique to increase the legibility of an output triangular set T is to reduce the polynomials in T by those with a constant initial.

We remarked in Section 3 that some superfluous components may occur in Kalkbrener's decompositions. There may exist indices i_1 and i_2 such that $\overline{\mathbf{W}(T_{i_1})} \supseteq \overline{\mathbf{W}(T_{i_2})}$. As suggested by Kalkbrener, one way to avoid some of these redundant computations is to use Krull's *Primeidealkettensatz* (see Samuel and Zariski, 1967, p.240): since the height of the saturated ideal of a regular chain T equals the number of its elements, we can delete from the output any regular chain which contains more elements than the input system. However, it is not sufficient to remove all superfluous components and we also use the following trick. If the initials of the polynomials in T_{i_1} are all in $\mathbf{k}$ then we have $\overline{\mathbf{W}(T_{i_1})} = \mathbf{V}(T_{i_1})$. In this case we may easily check the inclusion by using the algorithm split presented in Kalkbrener (1998), and remove eventually T_{i_2} from the output. With this last strategy we can remove eight components on the *Gonnet* example and four on *Butcher*. It happens that this technique slows down the computation. However, we use it with every example.

Most of the computing time in the methods of Kalkbrener and Lazard is dedicated to gcd computations modulo regular chains. Moreover, some of these gcd computations may be repeated. So we keep the results of these computations in hash-tables.

A last optimization of our implementation of Kalkbrener's method is the use of subresultant techniques for computing gcd modulo regular chains. This limits the growth of the coefficients. Note that it is also possible to construct normalized triangular sets in Kalkbrener's method as described in Aubry (1999).

The normalization and the computation of the square-free part of a polynomial w.r.t. a Lazard set are expensive operations. Thus, an improvement of Lazard's method is used to avoid these operations as much as possible. A typical situation where normalization can be avoided is the following. Assume that $T \subseteq \mathbf{P}_i$ is a Lazard set with i polynomials (thus the saturated ideal of T is zero-dimensional) and let p be a polynomial in $\mathbf{P}_i$. Recall that $\mathsf{normalize}(p, T)$ returns a list of couples (n_i, T_i) such that n_i is a normalized polynomial w.r.t. the Lazard set T_i and such that n_i and p are associated w.r.t. the t.o.s.e. associated with T_i. It follows from our assumptions that each n_i is a constant. Thus, either p is invertible w.r.t. T_i and we can set n_i to 1 or p and n_i are null. Hence, it is sufficient to check whether p is invertible w.r.t. T by a simple gcd computation without Bezout coefficients.

Another improvement of Lazard's method is to relax the square-freeness condition for Lazard sets in the intermediate computations. In fact, it is sufficient to guarantee that this property holds for the normalized sets of the final decomposition. The algorithm sketched in the previous section remains correct with this relaxation technique. However, the procedure for deciding whether a $\mathbf{W}(T_{i_1})$ is contained in a $\mathbf{W}(T_{i_2})$ becomes a partial operation: in some cases the inclusion cannot be checked. Thus, some redundant computations may only be removed at the end of the computations, when square-freeness is required. Note also that the normalization procedure needs some adaptation. This relaxation technique leads to a great speed up (except for the L_3 example). This is due to the fact that in practice very few normalized sets are not square-free.

Many other tricks can be used for improving Lazard's method, especially in dimension zero. For instance, it is possible to delay the normalization condition for zero-dimensional Lazard sets until the final decomposition. This implies some adaptation of the algorithm outlined in the previous section but speeds up the computations. In positive dimension, the normalization condition cannot be delayed. This explains why Lazard's method (using these relaxation techniques) is more satisfactory in dimension zero, as we will see in the next section.

As suggested in Lazard (1991), a last idea to speed up the computations of decompositions into Lazard triangular sets is to generate these chains by decreasing dimension. If this is done, the superfluous sets may be removed before any computation is performed with them. Since the practical complexity of gcd computations (and thus normalizations) w.r.t. a tower of simple extensions increases with the height of this tower, some unnecessary and expensive computations can be avoided by using this dimension argument.

5. Results

We now present our experimental results and give the triangular decompositions for some polynomial systems. The sources of our examples are specified in Table 5. For every example F (given as a polynomial set) and every method which decomposes $\mathbf{V}(F)$ into triangular systems $\sigma_1, \ldots, \sigma_r$ we give two pieces of information. The first one is the computing time (evaluation and garbage collector). It can be found in Table 1 for zero-dimensional examples and in Table 3 for examples of positive dimension. If $\mathbf{V}(F)$ has dimension zero, the second information is given in Table 2. It is a sequence $n(\sigma_1), \ldots, n(\sigma_r)$, where $n(\sigma_i)$ denotes the number of solutions of σ_i (counted with their multiplicities). For examples of positive dimension, the second information consists of a

Table 1. Timings for zero-dimensional examples.

	Wang	Wu	Kalkb.	Lazard	FGLM-AXIOM	FGLM-GB	HOM-GB
Trinks-1	<1	2	<1	1	1	<1	<1
Trinks-2	<1	<1	<1	<1	<1	<1	<1
Katsura-3	<1	<1	<1	1	<1	<1	<1
Katsura-4	3	>1000	5	7	2	<1	<1
Rose	2	>1000	3	22	>1000	65	5
S_4	69	>1000	51	66	70	22	3
S_5	74	>1000	200	495	280	84	11
Caprasse	7	>1000	2	2	8	<1	<1
Caprasse–Li	4	>1000	2	2	9	<1	<1
Arn-Laz	5	72	3	6	2	<1	<1
Cyclic-5	>1000	>1000	6	6	15	<1	<1
Wang-16	>1000	>1000	83	208	21	7	3
Singular	<1	1	<1	<1	<1	<1	<1
R_5	<1	>1000	<1	<1	43	<1	<1
R_6	<1	>1000	5	<1	>1000	66	<1
R_7	<1	>1000	>1000	<1	>1000	>1000	<1
L_2	<1	<1	<1	<1	<1	<1	<1
L_3	5	>1000	6	16	26	1	<1

Table 2. Degrees of outputs (zero-dimensional examples).

	Wang	Wu	Kalkbrener	Lazard	Gröbner basis
Trinks-1	10	10	10	10	10
Trinks-2	2	2	2	2	2
Katsura-3	8	8	1, 7	1, 7	8
Katsura-4	4, 12	?	4, 12	$2^2, 12$	16
Rose	132	?	4, 128	4, 128	136
S_4	$2^2, 39$	?	$2^2, 39$	$2^2, 39$	99
S_5	$2^2, 45$	?	$2^2, 45$	$2^2, 45$	99
Caprasse	$2^2, 4, 16^2$	?	$2, 6, 8, 16$	$4^2, 8, 16$	56
Caprasse–Li	$2, 6, 8, 12^2$	?	$4^2, 8, 16$	$4^2, 8, 16$	56
Arn-Laz	$2, 18$	$2, 18$	$2, 18$	$2, 18$	20
Cyclic-5	?	?	$10^3, 20^2$	$10^5, 20$	70
Wang-16	?	?	24	24	24
Singular	4^2	$1, 3, 4$	$1, 3, 4$	$1^2, 2, 4$	8
R_5	$1, 120$	?	$1, 120$	$1, 120$	121
R_6	1, 720	?	1, 720	1, 720	721
R_7	1, 5040	?	1, 5040	1, 5040	5041
L_2	$2, 3$	$2, 3$	$1^3, 2$	$1^3, 2$	8
L_3	$1, 2, 6, 8, 18, 32^2$	?	$1^5, 2^4, 8, 12, 16, 32$	$1^5, 2^4, 8, 12, 16, 32$	81

sequence $d(\sigma_1), \ldots, d(\sigma_r)$, where $d(\sigma_i)$ denotes the dimension of $\mathsf{sat}_n(\sigma_i)$. It is given in Table 4.

In order to make these sequences of numbers easy to read, we use some notations. Let us take the *Caprasse* example with Wang's method in Table 2. The sequence $2^2, 4, 16^2$ means that the decomposition contains two triangular sets with 16 solutions, one triangular set with four solutions and two triangular sets with two solutions. The same kind of notation applies for sequences of dimensions.

During our checking process, the empty components produced by Wu's method are automatically removed. So their number is not reported in our tables. In a similar way, the degrees given for Wu's method are obtained after removing the empty components. Note that the number of empty components depends on the heuristics and optimizations used in implementing Wu's method. So counting them does not make sense. Without using this *cleaning process* the decompositions of Wu's method are not easy to read, even for small

Table 3. Timings for positive dimension examples.

	Wang	Wu	Kalkb.	Lazard	SUGAR-AXIOM	SUGAR-GB	HOM-GB
DoTr	<1	<1	1	<1	3	<1	<1
Alonso	<1	<1	<1	<1	<1	<1	<1
Alonso–Li	9	>1000	6	>1000	1	<1	<1
Wang-91c	1	4	2	3	1	<1	<1
Wu-87b	<1	<1	<1	<1	<1	<1	<1
Wang-2	4	53	51	177	9	<1	<1
Wang-12	3	10	<1	2	<1	<1	<1
Gerdt	2	16	5	9	3	<1	<1
Gonnet	1	3	<1	5	<1	<1	1
Butcher	4	4	3	1	300	46	<1
Hairer-1	<1	<1	<1	<1	<1	<1	<1
Hairer-2	4	>1000	7	74	>1000	>1000	33
Vermeer	>1000	>1000	8	>1000	>1000	>1000	<1
Neural	>1000	>1000	8	>1000	8	<1	<1
Liu	95	>1000	66	>1000	>1000	>1000	2
Liu–Li	48	>1000	108	>1000	>1000	>1000	1
Romin	2	8	9	8	2	<1	<1
f-633	3	32	2	50	12	<1	1

Table 4. Dimensions for positive dimension examples.

	Wang	Wu	Kalkbrener	Lazard
DoTr	$0^2, 1$	$0^2, 1$	1	$0^2, 1$
Alonso	$2^2, 3$	$2^2, 3$	3	$2^2, 3$
Alonso–Li	$1^7, 2^3, 3$	?	3	?
Wang-91c	1^3	1^4	1^3	1^4
Wu-87b	$0, 1$	$0^5, 1$	1	$0^5, 1$
Wang-2	$1^4, 2$	$1^5, 2$	$1^8, 2$	$1^4, 2$
Wang-12	$1^9, 2^2$	$1^5, 2^2$	$1, 2^2$	$1, 2^2$
Gerdt	$1^2, 2^3, 3$	$1^7, 2^5, 3$	$1^3, 2^3, 3$	$1^2, 2^3, 3$
Gonnet	3^3	3^3	3^3	3^3
Butcher	$0, 2^2, 3^4$	$0, 2^5, 3^3$	$0, 2^2, 3^3$	$0, 2^2, 3^3$
Hairer-1	$1^2, 2^3$	$1^4, 2^3$	2^3	$1^2, 2^3$
Hairer-2	$1^3, 2$	?	2	$1^3, 2$
Vermeer	?	?	1^2	?
Neural	?	?	1^2	$0^{16}, 1^2$
Liu	$0^9, 1$	?	1	?
Liu–Li	$0^6, 1$	?	1	?
Romin	$3^5, 4^5, 5$	$3^6, 4^5, 5$	5	$3^4, 4^5, 5$
f-633	$1^3, 2$	$0^3, 1^{10}, 2$	2	$0, 1^4, 2$

examples. See, for instance, the output of the *Singular Points* example. Another example is the *Romin* system. Our implementation of Wu's method produces 31 components, 17 being empty and four being non-empty but redundant. Our implementation of Wang's method may also produce empty components. However, it happens much less frequently than with that of Wu.

We also give the timings for computing the lexicographical Gröbner bases with AXIOM and GB. To compute these bases for zero-dimensional systems we use the *FGLM* algorithm (Faugère *et al.*, 1993) with AXIOM and GB. For systems of positive dimension we use the *Sugar* algorithm (Giovinni *et al.*, 1991) with AXIOM and GB too.

The AXIOM timings for lexicographical Gröbner bases are interesting to give an idea on the efficiency of both classes of methods, based on different tools (triangular systems and Gröbner bases), but implemented within similar conditions (using the same AXIOM polynomial domain). Nevertheless, remember that solving algebraic systems does not

Table 5. Examples.

Example	Source or description
Trinks-1	(Boege *et al.*, 1986) with $B < S < T < Z < P < W$.
Trinks-2	(Boege *et al.*, 1986) with $B < S < T < Z < P < W$.
Katsura-3	(Boege *et al.*, 1986) with $U3 < U2 < U1 < U0$.
Katsura-4	(Boege *et al.*, 1986) with $U0 < U1 < U2 < U3 < U4$.
Rose	(Boege *et al.*, 1986) with $A46 < U4 < U3$.
S_4	4-body problem (Kotsireas, 1998) with $\psi < s < p$.
S_5	5-body problem (Kotsireas, 1998) with $\psi < s < p$.
Caprasse	$\{(2ty-2)x+zy^2-z, (t^2-1)x+2tzy-2z, -zx^3+(4ty+4)x^2+(4zy^2+4z)x+2ty^3-10y^2-10ty+2, (-z^3+(4t^2+4)z)x+(4tz^2+2t^3-10t)y+4z^2-10t^2+2\}$ with $t < z < y < x$.
Caprasse–Li	Caprasse with $x < y < z < t$ as in Li (1995).
Arn-Laz	Arnborg–Lazard (Giovinni *et al.*, 1991) with $x < y < z$.
Cyclic–5	(Lazard, 1992) with $e < d < c < b < a$.
Wang-16	problem 5(a) in Czapor and Geddes (1986) with $d < p < c < q$ and example 16 in Wang (1993b).
Singular	$\{f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\}$ where $f = (y-x)(y^2+x^2-1)(y^2-x)$ and $x < y$.
R_5	$\{x_1(x_1+1), (x_2^2+x_2+1)x_1+x_2, p_3, p_4, p_5\}$ where $p_i = x_i x_{i-1}^{i-1} + (x_i^i+1)x_{i-1} + x_i$ and $x_5 < \cdots < x_1$.
R_6	$\{x_1(x_1+1), (x_2^2+x_2+1)x_1+x_2, p_3, p_4, p_5, p_6\}$ with $x_6 < \cdots < x_1$.
R_7	$\{x_1(x_1+1), (x_2^2+x_2+1)x_1+x_2, p_3, p_4, p_5, p_6, p_7\}$ with $x_7 < \cdots < x_1$.
L_2	$\{x_1^2+x_2+x_3-1, x_1+x_2^2+x_3-1,\ x_1+x_2+x_3^2-1\}$ with $x_1 < x_2 < x_3$.
L_3	$\{x_1^3+x_2+x_3+x_4-1, x_1+x_2^3+x_3+x_4-1, x_1+x_2+x_3^3+x_4-1, x_1+x_2+x_3+x_4^3-1\}$ with $x_1 < \cdots < x_4$.
Do-Tr	(Donati and Traverso, 1989).
Alonso	$\{(u-r-2)x-u-2t^4+1, ry-t^2u-1, tuz-1, v-tr\}$ with $t < r < u < v < z < y < x$.
Alonso–Li	Alonso with $v < z < y < x < t < u < r$ as in Li (1995).
Wang-91c	Wang's example in Wang (1995).
Wu-87b	Wu's example in Wang (1995).
Wang-2	(Wu, 1986) ex. 4 with $x_{10} < \cdots < x_i < x_{i+1} < \cdots < x_{105}$.
Wang-12	(Bronstein, 1986) p. 248 with $a < b < y < x$ and example 12 in Wang (1993b).
Gerdt	(Boege *et al.*, 1986) with $L1 < L2 < \cdots < L7$.
Gonnet	(Boege *et al.*, 1986) with $A0 < A2 \ldots < A5 < B0 \ldots < B5 < C0 \ldots < C5$.
Butcher	(Boege *et al.*, 1986) with $B < C2 < C3 < A < B3 < B2 < A32 < B1$.
Hairer-1	(Boege *et al.*, 1986).
Hairer-2	(Boege *et al.*, 1986) with $A43 > A42 > A41 > A32 > A31 > A21 > B1 > B2 > B3 > B4 > C4 > C3 > C2$.
Vermeer	(Wang, 1993b) ex.2 p.110.
Neural	Neural Network (Kalkbrener, 1991).
Liu	(Liu, 1989) with $a < t < z < y < x$.
Liu–Li	Liu with $a < x < y < z < t$ as in Li (1995).
Romin	Robot ROMIN (González-López and Recio, 1993) $\{-ds_1-a, dc_1-b, l_2c_2+l_3c_3-d, 2s_2+l_3s_3-c, s_1^2+c_1^2-1, s_2^2+c_2^2-1, s_3^2+c_3^2-1\}$ with $d < c < b < a < l_3 < l_2 < c_3 < s_3 < c_2 < s_2 < c_1 < s_1$.
f-633	$\{2u_6+2u_5+2u_4+2u_3+2u_2+1, 2U_6+2U_5+2U_4+2U_3+2U_2+1, (8u_6+8u_5)U_5+(8u_6+8u_5+8u_4)U_4+(8u_6+8u_5+8u_4+8u_3)U_3+(8u_6+8u_5+8u_4+8u_3+8u_2)U_2-13, (8u_5+8u_4+8u_3+8u_2)U_6+(8u_5+8u_4+8u_3+8u_2)U_5+(8u_4+8u_3+8u_2)U_4+(8u_3+8u_2)U_3+8u_2U_2-13, u_2U_2-1, u_3U_3-1, u_4U_4-1, u_5U_5-1, u_6U_6-1\}$ with $u_2 < u_3 < u_4 < u_5 < u_6 < U_2 < U_3 < U_4 < U_5 < U_6$ (Faugère *et al.*, 1998).

have the same meaning for those different classes of methods. One cannot extract the same information from a Gröbner basis as from a triangular decomposition.

Unfortunately, our AXIOM programs for computing triangular decompositions are probably far to be as optimized as some Gröbner bases implementations are. Thus, the ratio between AXIOM and GB timings for the *FLGM* and *Sugar* algorithms may inform us of the eventual increase of performances that we could obtain with optimized implementations. These GB timings should only be understood for this purpose and should not be considered as references. Indeed, several efficient algorithms and tools have been developed recently in the field of Gröbner bases such as the choice of good strategies (Giovinni *et al.*, 1991), new data representations or algorithms for changing monomial ordering (Faugère, 1994; Traverso, 1996; Collart *et al.*, 1997). These methods generally provide a good way to compute a lexicographical Gröbner basis for our examples when a direct computation is not efficient. Some of them are used by default in some software for computing lexicographical Gröbner bases (MAGMA for instance). Thus, we also used the following strategy for computing the lexicographical Gröbner basis with GB:

(1) Computing a Gröbner basis for the total degree ordering of the input system F.
(2) Homogenizing the obtained system.
(3) Computing the lexicographical Gröbner basis from this system.
(4) Dishomogenizing the result and reducing it since this is a Gröbner basis of $\langle F\rangle$ which is not generally minimal.
(5) Verifying the result with the Hilbert function (see Traverso, 1996).

With this last method, all the Gröbner bases of our tables can be computed with GB. The timings are given in the column HOM-GB of Tables 1 and 3. Note that for some examples (*Rose, Hairer-2*) our implementations of Wang's method and Kalkbrener's method give better timings than GB. Moreover, on several relevant examples (*S-5, Wang-16, Butcher, Vermeer, Liu*) the ratio between GB and our implementation of Kalkbrener's method is satisfactory for the latter. Indeed, it is well known that polynomial arithmetic can be implemented much more efficiently in C/C++ than in AXIOM.

The degree of the ideal generated by the input system F may be obtained from a Gröbner basis. We mention it in Table 2 (column *Gröbner basis*) for the zero-dimensional examples. The existence of multiplicities thus appears by comparing this degree with the sum of the degrees given in the column *Lazard* since the square-free triangular sets produced by Lazard's method generate radical ideals. By comparing the decompositions produced by our implementations of the methods of Lazard and Kalkbrener, we observed that the saturated ideals of the regular chains of the latter method are generally radical. Let us specify finally that the dimension of an ideal $\langle F\rangle$ is equal to the maximal dimension occuring in any of its decompositions reported in Table 4.

All our benchmarks have been made three times. First on a SPARC 10 station then on Pentium II PC and finally on an DEC Alpha station. The timings reported here are obtained with this last architecture, with a EV5.6 (21164A) processor operating at 531 MHz. This machine has 512 Meg of RAM memory and runs under OSF1(V4).

These timings are given in seconds. In the tables <1 means that the computation finished within less than a second whereas >1000 means that the computation was not finished after 1000 seconds. This last notation does not mean that the computation never finished. For instance, our implementation of Kalkbrener's method produced an output with the R_7 example within 1672 seconds and our implementation of Lazard's method produced an output with the *Neural* example within 5160 seconds.

Example 5.1. (Donati-Traverso) This classic example shows that the computation of a lexicographical Gröbner basis is not always very interesting. The computation with AXIOM is less efficient than the triangular decompositions and the Gröbner basis is much bigger than the results obtained by these triangular decompositions.

Example 5.2. (Cyclic-5) This other classic example is not considered as difficult for Gröbner bases techniques. The results show the interest of methods based on gcd computations modulo towers of extensions, since they are the only ones which compute a triangular decomposition. Their computations are easier than the computation of the lexicographical Gröbner basis with AXIOM, and they detect more splits than lexTriangular. Some coefficients in the output obtained with Kalkbrener's method are bigger than the coefficients in the output of Lazard's method and lexTriangular. This difference is due to the fact that the latter methods produce normalized triangular sets whereas Kalkbrener's method does not. For $n > 5$, it is much more difficult to obtain a direct triangular decomposition for the cyclic-n system than computing a lexicographical Gröbner basis.

Example 5.3. (Systems R_n) All methods produce the same output (which can be computed by hand from the input system) but our implementation of Kalkbrener's method is inefficient in this particular example. This follows from the sense of reduction that we use for our regular chains. A polynomial in a regular chain T is reduced only w.r.t. the other polynomials in T whose initials are in the base field $\mathbf{k}$. In many cases the output is legible enough with this choice. But for the systems R_n the full reduction (as it is used in Lazard's method and lexTriangular) simplifies a lot the intermediate triangular sets and thus speeds up the gcd computations. If we change the notion of reduction in our implementation of Kalkbrener's method, then the example R_7 is performed within less than a second. However, with this choice, we reduce the performances of this method on several examples.

Example 5.4. (Neural) In our implementation, only the methods based on gcd computations over towers of simple extensions succeed with this example. The decomposition computed by our implementation of Kalkbrener's method is much more compact than the lexicographical Gröbner basis and it is obtained within the same timing. It produces two regular chains corresponding to saturated ideals of dimension one. The decomposition computed by our implementations of Lazard's method produce 18 Lazard sets, two of them corresponding to components of dimension one and the others to zero-dimensional components. The use of normalization and full reduction in this method dramatically slows down the intermediate computations with this example. Moreover, this example illustrates the fact that solving in the sense of the regular zeroes (as in the methods of Wu, Wang and Lazard) is sometimes much harder than solving in the sense of the Zariski closure (as in Kalkbrener's method).

Example 5.5. (Singular points) Here the methods give different results and we note that Lazard's decomposition is more legible than the others. One can check that $\mathbf{W}(W_2) = \mathbf{W}(C_3)$ and $\mathbf{W}(W_3) = \mathbf{W}(C_2)$.

Wang's method:

$$\begin{aligned}
&(T_1 = \{x^2 + x - 1, xy^2 + x - 1\},\\
&Q_1 = \emptyset),\\
&(T_2 = \{2x^4 - 2x^3 - x^2 + x,\\
&\qquad (46x^5 + 48x^4 - 64x^3 - 24x^2 + 18x + 2)y +\\
&\qquad -48x^5 - 51x^4 + 70x^3 + 28x^2 - 25x\},\\
&Q_2 = \emptyset),
\end{aligned}$$

Wu's method:

$$\begin{aligned}
W_1 &= \{x^2 + x - 1, (86x - 53)y^2 + 139x - 86\},\\
W_2 &= \{2x^4 + x^3 - 3x^2 + x,\\
&\quad (208x^3 + 104x^2 - 312x + 104)y +\\
&\quad 11960x^9 - 5660x^8 - 20990x^7 + 26143x^6 +\\
&\quad -9968x^5 - 990x^4 + 1601x^3 - 218x^2 - 36x\},\\
W_3 &= \{4x^7 - 2x^6 - 10x^5 + 9x^4 + 2x^3 - 4x^2 + x,\\
&\quad (66x^6 + 29x^5 - 131x^4 + 24x^3 + 43x^2 - 15x)y +\\
&\quad -64x^{10} - 76x^9 + 142x^8 + 96x^7 +\\
&\quad -124x^6 - 21x^5 + 32x^4 + 2x^3 - 3x^2\}.
\end{aligned}$$

Kalkbrener's method:

$$\begin{aligned}
C_1 &= \{x^2 + x - 1, (86x - 53)y^2 + 139x - 86\},\\
C_2 &= \{2x^3 - 2x^2 - x + 1, (102x^2 - 39x - 23)y - 63x^2 - 28x + 51\},\\
C_3 &= \{x, y\}.
\end{aligned}$$

Lazard's method:

$$\begin{aligned}
L_1 &= \{2x^2 - 1, y - x\},\\
L_2 &= \{x^2 + x - 1, y^2 - x\},\\
L_3 &= \{x, y\},\\
L_4 &= \{x - 1, y - 1\}.
\end{aligned}$$

Example 5.6. (Caprasse) Unfortunately, our implementation of Wu's method does not succeed in this example. The methods based on gcd computations provide the best output and the best timings. Moreover, this example shows that these methods may provide a better way to get a triangular decomposition than a Gröbner basis computation followed by a call to the lexTriangular algorithm.

Wang's method:

$$\begin{aligned}
&(T_1 = \{t^4 - 10t^2 + 1, z, (t^3 - 5t)y - 5t^2 + 1, x\},\\
&Q_1 = \emptyset),\\
&(T_2 = \{3t^4 + 10t^2 + 3,\\
&\qquad (44289t^2 + 14769)z^4 + (-354288t^2 - 118080)z^2
\end{aligned}$$

$$
\begin{aligned}
&\qquad +708592t^2 + 236208,\\
&\qquad ((354276t^3 + 118044t)z^2 + 708600t^3 + 236232t)y\\
&\qquad +(-44289t^2 - 14769)z^4 + (-708588t^2 - 236196)z^2 +\\
&\qquad -2834360t^2 - 944808,\\
&\qquad (t^2 - 1)x + 2tzy - 2z\},\\
&Q_2 = [(29523t^2 + 9837)z^2 + 59050t^2 + 19686]),\\
&(T_3 = \{t^2 + 1, z, ty - 1, x\},\\
&Q_3 = \emptyset),\\
&(T_4 = \{t^2 - 1, z^8 - 16z^6 + 256z^2 - 256, ty - 1,\\
&\qquad (z^3 - 8z)x + (-4tz^2 + 8t)y - 4z^2 + 8\},\\
&Q_4 = \emptyset),\\
&(T_5 = \{t^2 - 1, z, ty + 1, x\},\\
&Q_5 = \emptyset).
\end{aligned}
$$

Kalkbrener's method:

$$
\begin{aligned}
C_1 &= \{t^2 - 1, z^8 - 16z^6 + 256z^2 - 256, ty - 1,\\
&\qquad (z^3 - 8z)x + (4tz^2 + 8t)y - 12z^2 + 8\},\\
C_2 &= \{3t^4 + 10t^2 + 3, 3t^2z^2 - 14t^2 - 6,\\
&\qquad (z^2 + 5t^2 - 1)y - tz^2 + t^3 - 5t,\\
&\qquad (t^2 - 1)x + 2tzy - 2z\},\\
C_3 &= \{t^2 - 1, z, ty + 1, x\},\\
C_4 &= \{t^6 - 9t^4 - 9t^2 + 1, z, (t^3 - 5t)y - 5t^2 + 1, x\}.
\end{aligned}
$$

Lazard's method:

$$
\begin{aligned}
L_1 &= \{t^2 - 1, z^8 - 16z^6 + 256z^2 - 256, y - t,\\
&\qquad 96x - z^7 + 14z^5 + 16z^3 - 128z\},\\
L_2 &= \{3t^2 + 1, 3z^2 + 4, y + t, x + z\},\\
L_3 &= \{t^2 + 3, z^2 - 4, y + t, x - z\},\\
L_4 &= \{t^8 - 10t^6 + 10t^2 - 1, z, 24y - 5t^7 + 49t^5 + 5t^3 - 25t, x\}.
\end{aligned}
$$

Example 5.7. (Alonso) Alonso's example corresponds to a prime ideal of dimension three. Thus, Kalkbrener's method describes it with only one regular chain C_1. The other algorithms extract points which are in the closure of the regular zeroes of C and provide similar results.

Wu's method:

$$
\begin{aligned}
W_1 &= \{r, t^2u + 1, v, tz + t^2, (2t^2 + 1)x + t^2u + 2t^6 - t^2\},\\
W_2 &= \{r + 2t^4 + 1, u - r - 2, v - tr,\\
&\qquad (2t^5 - t)z + 1, (2t^4 + 1)y + t^2u + 1\},\\
W_3 &= \{v - tr, tuz - 1, ry - t^2u - 1(u - r - 2)x - u - 2t^4 + 1\}.
\end{aligned}
$$

Wang's method:

$$\begin{aligned}
&(T_1 = \{v - tr, tuz - 1, ry - t^2u - 1, (u - r - 2)x - u - 2t^4 + 1\},\\
&Q_1 = [u, r, t, u - r - 2]),\\
&(T_2 = \{r, t^2u + 1, v, z + t, (2t^2 + 1)x + t^2u + 2t^6 - t^2\},\\
&Q_2 = [2t^2 + 1, t]),\\
&(T_3 = \{r + 2t^4 + 1, u + 2t^4 - 1, v + 2t^5 + t, (2t^5 - t)z + 1\\
&\qquad (2t^4 + 1)y - 2t^6 + t^2 + 1\},\\
&Q_3 = [2t^4 + 1, 2t^4 - 1, t]).
\end{aligned}$$

Kalkbrener's method:

$$C_1 = \{v - tr, tuz - 1, ry - t^2u - 1, (u - r - 2)x - u - 2t^4 + 1\}.$$

Lazard's method:

$$\begin{aligned}
&L_1 = \{v - tr, tuz - 1, ry - t^2u - 1, (u - r - 2)x - u - 2t^4 + 1\},\\
&L_2 = \{r + 2t^4 + 1, u + 2t^4 - 1, v + 2t^5 + t, (2t^5 - t)z + 1,\\
&\qquad (2t^4 + 1)y - 2t^6 + t^2 + 1\},\\
&L_3 = \{r, t^2u + 1, v, z + t, (2t^2 + 1)x + 2t^6 - t^2 - 1\}.
\end{aligned}$$

6. Some Other Implementations

The main purpose of our work is a comparison within a common environment of four triangulation methods based on pseudo-division. We discussed this point in Section 4. However, the problem of solving polynomial systems by means of triangular sets has been studied by other approaches. Indeed, triangulation algorithms may involve techniques (factorization into irreducibles, Gröbner basis computations, dynamic evaluation) or data structures that we do not use. In this section, we take into account some experimental results from the literature and try to compare them with our work.

Lazard (1992) proposed an algorithm (called *Lextriangular*) to compute triangular decompositions of zero-dimensional ideals. This algorithm is based on a structure theorem for zero-dimensional lexicographical Gröbner bases (Gianni, 1987; Kalkbrener, 1987) and requires such a basis as input. Moreno Maza and Rioboo (1995) reported an efficient implementation in AXIOM of the *Lextriangular* algorithm. Their work is based on an enhanced version of the sub-resultant algorithms and their AXIOM program can process the *Cyclic-7* system (the lexicographical Gröbner basis being computed by GB). However, recall that this algorithm can only be applied to zero-dimensional lexicographical Gröbner bases, whereas the four methods described in this paper can be applied on any polynomial system. Moreover, it happens, with some hard zero-dimensional examples, that using one of these four methods provides a quicker answer than by computing a lexicographical Gröbner basis.

Möller (1993) presents another algorithm which decomposes zero-dimensional varieties from Gröbner bases. It involves lexicographical Gröbner basis computations of quotient ideals which are generally expensive. This approach has been improved and extended to positive dimension in Gräbe (1995) where an implementation in REDUCE is reported. Factorization is also a major tool in these methods. We explained in the introduction that we wanted to avoid Gröbner basis computations for complexity considerations. However, it is a personal choice and it remains interesting for further investigations to compare experimental factorization based methods with the methods studied in this paper. The algorithms presented in Gräbe (1995) rely on factorized Gröbner bases and reduction to dimension zero. Several zero-dimensional examples are given. The variant ZS2 from that paper for decomposing zero-dimensional varieties seems the most interesting. It uses as much as possible the *degree-rev-lex* ordering for computing Gröbner bases. This allows us to decompose the example *Katsura-5* which is not computed by our implementation. The other timings for this variant compare with those given by our implementation. However, note that the hardware and software conditions are different and that only five zero-dimensional examples (three of them being *Katsura-n* systems) are given. Some examples of positive dimension are also given in Gräbe (1995), such as *Donati–Traverso, Gonnet* and *Hairer-1*, which are computed efficiently by our implementation. We think that a significant comparison should involve investigations with more difficult examples.

Dynamic evaluation (Gómez-Díaz, 1994) provides a way to solve polynomial systems by means of triangular sets. It involves particular data structures like dynamic sets and dynamic polynomials. Therefore it could not be part of our comparative implementation. However, we run the implementation available in AXIOM on our examples. It appears that dynamic evaluation is much slower than our implementations of the four methods, but recall it is designed to deal with more general subjects than polynomial system solving.

Li (1995) reported an efficient implementation of the characteristic set method of Wu, based on SACLIB 1.2. This implementation involves factorization, sub-resultant techniques and modular algorithms. Li pointed out that it was a difficult task to check the correctness of the triangular decompositions. However, he did not report on the size and the legibility of his decompositions. Li's implementation has good timings for the *Vermeer* and the *Alonso–Li* examples. His implementation can process the *Cyclic-6* system whereas none of the implementations of the four methods can. However, his implementation fails on the *Liu* example.

Wang (1996) reports experiments with the methods of Wu and Wang. He concludes that the different variants of Wu's method are less efficient than the method developed by Wang. These implementations do not use the same notion of reduction as we do for computing characteristic sets. Nevertheless, our experiments lead to the same conclusions.

Many of our examples are given by Wang (1996). However, we do not compare our implementation of Wang's method with the author's since our implementation techniques and experimental conditions are different.

In Moreno Maza (1998), a new algorithm for solving polynomial systems by means of regular chains is reported. This algorithm is able to produce triangular decompositions either in the sense of the Zariski closure (as in Kalkbrener's method) or in the sense of the regular zeroes (as in Lazard's method). Moreover, it allows to use either (general) regular chains, or square-free regular chains, or Lazard sets.

Thus, this algorithm allows to measure experimentally the cost of a given property for a triangular decomposition. It has been implemented and tested with the same hardware and software as our implementations of the four methods. We give here the timings for our most difficult examples of positive dimension. In the Table 6, the column $\mathbf{W}$ (resp. $\overline{\mathbf{W}}$) corresponds to decompositions in the regular zeroes sense (resp. in the Zariski closure sense) by means of (general) regular chains.

Table 6. Timings for Moreno Maza's method.

	$\overline{\mathbf{W}}$	**W**
Alonso–Li	6	>1000
Wang-2	34	35
Butcher	3	3
Hairer-2	16	30
Vermeer	3	7
Neural	3	14
Liu	11	132
Liu–Li	6	>1000

These results show that the cost of the property *solving in the sense of regular zeroes* is sometimes very high. This helps to understand the difference in performances between the methods of Kalkbrener and Lazard in positive dimension.

Improvements to Kalkbrener's method are reported in Aubry (1999). Several of our test examples are more efficiently performed with some optimizations. The implementation is realized in AXIOM. The basic idea of the algorithm decompose given in Kalkbrener (1998) is kept but stronger notions of regular chains may also be used for computing a triangular decomposition in the sense of Zariski closure. Thus, it is possible to produce square-free and normalized regular chains with efficiency. These extensions allows to compute the lexicographical Gröbner basis of ideals of relations in Galois Theory with timings comparable to the function Groebner in MAGMA. These Gröbner bases of zero-dimensional ideals are also Lazard sets.

7. Conclusions

Our conclusions are highly influenced by the way that we have implemented these four methods, even if we have tried to limit the differences which are due only to the implementation.

For easy examples, we remark that all methods generally have good computing times and that the legibility of the outputs that they produce is satisfactory. Nevertheless, Wu's method fails in some rather easy zero-dimensional examples (*Caprasse, R-5)* and for both cases of dimension zero and positive dimension, this method clearly solves less problems than the other methods. Moreover, for the most difficult examples that Wu's method can solve, the outputs are hard to read (for instance with the *Romin* example). In our opinion, the reason is the following. Wu's method cannot split the computations (in order to obtain several triangular sets) before computing a characteristic set of F (which is sometimes hard to compute, especially for zero-dimensional problems), whereas the other methods may split their computations earlier. More generally, it seems that methods based on gcd computations over a tower of simple extensions, namely those of Lazard and Kalkbrener, may discover factorizations that other methods cannot find (*Cyclic-5*).

Let us now concentrate on Wang's method. This method may be very efficient for difficult examples. Indeed, it has the best timings for the examples S_5, *Liu–Li*. However, the decompositions that it produces are generally less legible than the ones of Kalkbrener and Lazard. Furthermore, as Wu's method, the method of Wang is disappointing in some easy examples, namely *Caprasse, Cyclic-5, Neural.*

Kalkbrener's method is the only method which solves every example and often produces the most concise outputs (except for *Cyclic-5*). Moreover, some examples (*Vermeer,* *Neural*) can only be processed by this method. But one has to keep in mind that this method solves polynomial systems in a *more lazy way* than the other three in the case of positive dimension. Our strategy for this method is also inefficient for some zero-dimensional examples (*R-7*), whereas the methods of Lazard and Wang succeed with these examples. We think that the use of normalized triangular sets in Lazard's method is generally a good way to solve zero-dimensional problems. This may replace a large algebraic expression by a single integer.

However, normalization and square-free factorization over towers of separable extensions are time consuming. This is the reason why Lazard's method may also be inefficient in some easy examples (*Neural*). For describing affine varieties by means of regular zeroes of triangular sets, Lazard's method gives the best output. Moreover, this is the only method which produces decompositions without redundant components.

We think that the methods based on gcd computations over towers of extensions are promising. The experiments show that they must be investigated further for more efficiency. For the most difficult problems of positive dimension, it appears that solving in the sense of the Zariski closure (as in Kalkbrener's method) may be a good way to obtain a decomposition. Such a decomposition is sufficient when one is only interested in a description of the radical ideal associated with the input system. However, more investigations are needed for solving algebraic systems of positive dimension in the sense of the regular zeroes by means of Lazard triangular sets. Indeed, the computations of normalized gcds may be prohibitive in that case. We finally remark that for solving in the sense of the regular zeroes, Wang's method is a powerful tool in some hard examples.

Acknowledgements

We would like to thank Dongming Wang, Mike Dewar and the referees for their helpful suggestions on an earlier version of this paper. The second author is also grateful to the European research project FRISCO[†] for supporting his work.

References

Aubry, P. (1999). Ensembles triangulaires de polynômes et résolution des systèmes d'équations algébriques. Ph.D. Thesis, Université Paris 6.

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

Boege, W., Gebauer, R., Kredel, H. (1986). Some examples for solving systems of algebraic equations by calculating Gröbner bases. *J. Symb. Comput.*, **2**, 83–98.

Broadbery, P. A., Dooley, S. S., Iglio, P., Morisson, S. C., Steinbach, J. M., Sutor, R. S., Watt, S. M. (1994). *AXIOM Library Compiler User Guide*, 1st edn. Oxford, U.K., NAG, The Numerical Algorithms Group Limited. AXIOM is a registered trade mark of NAG.

Bronstein, M. (1986). Gsolve: a faster algorithm for solving systems of algebraic equations. In Char, B. W., ed., *Proceedings SYMSAC'86*, pp. 247–249. New York, ACM Press.

Buchberger, B. (1965). Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal. Ph. D. Thesis, University of Innsbruck.

Chou, S. (1988). *Mechanical Geometry Theorem Proving*. Dordrecht, Reidel.

Chou, S., Gao, X. (1990). Ritt–Wu's decomposition algorithm and geometry theorem proving. In *Proceedings CADE-10*, *Kaiserslautern, Germany*, LNCS **449**, pp. 207–220, Springer Verlag.

Chou, S., Gao, X. (1992). Solving parametric algebraic systems. In *Proceedings ISAAC'92*, *Berkeley, CA*, pp. 335–341, ACM Press, New York.

Collart, S., Kalkbrener, M., Mall, D. (1997). Converting bases with the Gröbner walk. *J. Symb. Comput.*, **24**, 465–470.

[†]A Framework for Integrated Symbolic/Numeric Computation. Esprit Scheme Project No. 21 024

Cox, D., Little, J., O'Shea, D. (1992). *Ideals, Varieties, and Algorithms.* New York, Springer-Verlag.

Czapor, S., Geddes, K. (1986). On implementing Buchberger's algorithm for Gröbner bases. In *Proceedings SYMSAC'86*, pp. 425–440. Waterloo, B.W. Char.

Della Dora, J., Discrescenzo, C., Duval, D. (1985). About a new method method for computing in algebraic number fields. In *Proceedings EUROCAL 85*, volume 2, LNCS **204**, pp. 289–290. Springer-Verlag.

Donati, L., Traverso, C. (1989). Experimenting the Gröbner basis algorithm with the ALPI system. *Proceedings ISSAC'89*, pp. 192–198. New York, ACM Press.

European Commission (1992). *PoSSo—Polynomial System Solving Research Project.* Esprit Scheme Project No. 6846. Has been extended to the FRISCO project.

Faugère, J.-C. (1994). Résolution des systèmes d'équations algébriques. Ph. D. Thesis, Université Paris 6.

Faugère, J.-C., Gianni, P., Lazard, D., Mora, T. (1993). Efficient computation of zero-dimensional Gröbner bases by change of ordering. *J. Symb. Comput.*, **16**, 329–344.

Faugère, J.-C., Moreau de Saint-Martin, F., Rouillier, F. (1998). Design of nonseparable bidimensional wavelets and filter banks using Gröbner bases techniques. *IEEE SP Trans. Signal Process.*, **46**. Special Issue on Theory and Applications of Filter Banks and Wavelets.

Gallo, G., Mishra, B. (1990). Efficient algorithms and bounds for Wu–Ritt characteristic sets. *Proceedings MEGA'90, Lavorno, Italy*, Progress in Mathematics 94, pp. 119–142, Birkhaüser.

Gianni, P. (1987). In *Properties of Gröbner Basis under Specializations*, LNCS **378**, pp. 293–297. Springer.

Giovinni, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C. (1991). "One sugar cube, please" or selection strategies in the Buchberger algorithm. In *Proceedings ISSAC'91*, pp. 49–54. New York, ACM Press.

Gómez-Díaz, T. (1994). Quelques applications de l'évaluation dynamique. Ph. D. Thesis, Université de Limoges.

González-López, M., Recio, T. (1993). The ROMIN inverse geometric model and the dynamic evaluation method. In Cohen, A. M., ed., *Proceedings of the 1991 SCAFI Seminar, Computer Algebra in Industry.* New York, Wiley.

Gräbe, H.-G. (1995). Triangular systems and factorized Gröbner bases. In *Proceedings AAECC 11, Paris*, Cohen, G., Giusti, M. and Mora, T., eds, LNCS **948**, pp. 248–261, Springer Verlag.

Huynh, D. (1986). A superexponential lower bound for Gröbner bases and Church Rosser commutative thue systems. *Inf. Control*, **68**, 196–206.

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

Kalkbrener, M. (1987). In *Solving Systems of Algebraic Equations by using Gröbner Bases*, LNCS **378**, pp. 282–292. Springer.

Kalkbrener, M. (1991). Three contributions to elimination theory. Ph. D. Thesis, Johannes Kepler University, Linz.

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

Kalkbrener, M. (1998). Algorithmic properties of polynomial rings. *J. Symb. Comput.*, **26**, 525–581.

Kotsireas, I. (1998). Algorithmes de résolution de systèmes polynomiaux: application aux configurations centrales du problème des N corps en mécanique céleste. Ph. D. Thesis, Université Paris 6.

Lazard, D. (1991). A new method for solving algebraic systems of positive dimension. *Discrete Appl. Math.*, **33**, 147–160.

Lazard, D. (1992). Solving zero-dimensional algebraic systems. *J. Symb. Comput.*, **15**, 117–132.

Li, Z. (1995). An implementation of the characteristic set method for solving algebraic equations. In *Proceedings POSSO Workshop on Software, Paris, France*, Université Paris 6, pp. 107–122.

Liu, Z. (1989). An algorithm on finding all isolated zeros of polynomial equations. *MM Research Preprints*, **4**, 63–76.

Mayr, E., Meyer, A. (1982). The complexity of the word problems for commutative semigroups and ideals. *Adv. Math.*, **46**, 305–329.

Möller, H. M. (1993). On decomposing systems of polynomial equations with finitely many solutions. *Appl. Algebra Eng. Commun. Comput.*, **4** 217–230.

Moreno Maza, M. (1997). Calculs de pgcd au-dessus des tours d'extensions simples et résolution des systèmes d'équations algébriques. Ph. D. Thesis, Université Paris 6.

Moreno Maza, M. (1998). A new algorithm for computing triangular decompositions of algebraic varieties. Technical Report TR 4/98, NAG Ltd, Oxford, U.K.

Moreno Maza, M., Rioboo, R. (1995). Polynomial gcd computations over towers of algebraic extensions. *Proceedings AAECC-11*, pp. 365–382. Springer.

Ritt, J. F. (1932). *Differential Equations from an Algebraic Standpoint*, volume 14. New York, American Mathematical Society.

Ritt, J. F. (1966). *Differential Algebra.* New York, Dover.

Samuel, P., Zariski, O. (1967). *Commutative Algebra.* Princeton, NJ, Van Nostrand.

Traverso, C. (1996). Hilbert functions and the Buchberger algorithm. *J. Symb. Comput.*, **22**, 355–376.

Wang, D. M. (1991). On Wu's method for solving systems of algebraic equations. Technical Report RISC-LINZ Series no 91-52.0, Johannes Kepler University, Linz, Austria.

Wang, D. M. (1992). Some improvements on Wu's method for solving systems of algebraic equations. In Wu, W.-T. and Cheng, M.-D., eds, *Proceedings of the Int. Workshop on Math. Mechanisation, Beijing, China.* Institute of Systems Science, Academia Sinica.

Wang, D. M. (1993a). An elimination method based on Siedenberg's theory and its applications. In Eysette, F. and Galligo, A., eds, *Comptutational Algebraic Geometry*, pp. 301–328. Boston, Birkhaüser.

Wang, D. M. (1993b). An elimination method for polynomial systems. *J. Symb. Comput.*, **16**, 83–114.

Wang, D. M. (1995). An implementation of the characteristic set method in Maple. In Pfalzgraf, J. and Wang, D. M., eds, *Automated Practical Reasoning: Algebraic Approaches*, pp. 187–201. Wien, Springer.

Wang, D. M. (1996). Solving polynomial equations: characteristic sets and triangular systems. *Math. Comput. Simul.*, **42**, 339–345.

Wang, D. M. (1998). Decomposing polynomial systems into simple systems. *J. Symb. Comput.*, **25**, 295–314.

Wu, W. T. (1986). On zeros of algebraic equations—an application of Ritt principle. *Kexue Tongbao*, **31**, 1–5.

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

Yang, L., Zhang, J. (1994). Searching dependency between algebraic equations: an algorithm applied to automated reasoning. In Johnson, J., McKee, S. and Vella, A., eds, *Artificial Intelligence in Mathematics*, pp. 147–156. Oxford, Oxford University Press.